AD-A156 519

# NAVAL POSTGRADUATE SCHOOL
## Monterey, California

DTIC
ELECTE
JUL 11 1985
S D
G

# THESIS

MICROLAN

FILE TRANSFER PROGRAM FOR MICROPROCESSORS

by

Roger Dean Jaskot
and
Harold Wayne Henry

March 1985

Thesis Advisor: G.E. Latta

Approved for Public Release; Distribution Unlimited

DTIC FILE COPY

85 6 19 024

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. AD-A156519 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) Microlan File Transfer Program for Microprocessors | | 5. TYPE OF REPORT & PERIOD COVERED Master's Thesis March 1985 |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s) Roger Dean Jaskot<br>Harold Wayne Henry | | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School<br>Monterey, California 93943 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School<br>Monterey, California 93943 | | 12. REPORT DATE March 1985 |
| | | 13. NUMBER OF PAGES 155 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report) Unclassified |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)
Approved for Public Release; Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

| | |
|---|---|
| Microlan | FILE TRANSFER |
| LAN | Network |
| Local Area Network | Information Transfer |
| MICRO | Electronic Mail |
| MICROCOMPUTER | RS232 |

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
The age of automation has established its foothold in today's society. Computerization now affects almost everyone's job, and sharing of information is vital to successful job performance. Manual transfer of information is inefficient and prone to error, so another means is needed. One option is computer networking. Both Local Area Networks and long-haul networks presently exist, but they are either very expensive or hardware dependent.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S N 0102-LF-014-6601

Unclassified

 SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

It would normally require a long lead time and high costs for the military to acquire an information transfer system. To provide a readily available, low-cost file transfer system, the authors developed an assembly language program named MICROLAN, which is written to work with three of the main microcomputer operating systems (CP/M-80, CP/M-86, and MS.DOS) and to take advantage of RS232 technology. MICROLAN was tested successfully for file transfer at up to 4800 baud, and suggestions have been included as to possible uses for MICROLAN in the military environment. Additionally, possible methods for upgrading MICROLAN are also included.

| Accession For | |
|---|---|
| NTIS GRA&I | X |
| DTIC TAB | ☐ |
| Unannounced | ☐ |
| Justification | |
| By | |
| Distribution/ | |
| Availability Codes | |
| Dist | Avail and/or Special |
| A/1 | |

Approved for public release; distribution is unlimited.

# MICROLAN
# File Transfer Program for Microprocessors

by

Roger D. Jaskot
Lieutenant, United States Navy
B.S., Illinois Institute of Technology, 1979

and

Harold W. Henry
Captain, United States Air Force
B.S., Texas Tech University, 1975

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY
(Command, Control, and Communications)

from the

NAVAL POSTGRADUATE SCHOOL
March 1985

Authors: ______________________
Roger D. Jaskot

______________________
Harold W. Henry

Approved by: ______________________
G.E. Latta, Thesis Advisor

______________________
J.T. Malckas, Second Reader

______________________
M.G. Sovereign, Chairman,
Command, Control, and Communications Joint Academic Group

______________________
David A. Schrady,
Academic Dean

## ABSTRACT

The age of automation has established its foothold in today's society. Computerization now affects almost everyone's job, and sharing of information is vital to successful job performance. Manual transfer of information is inefficient and prone to error, so another means is needed. One option is computer networking. Both Local Area Networks and long-haul networks presently exist, but they are either very expensive or hardware dependent.

It would normally require a long lead time and high costs for the military to acquire an information transfer system. To provide a readily available, low-cost file transfer system, the authors developed an assembly language program named MICROLAN, which is written to work with three of the main microcomputer operating systems (CP/M-80, CP/M-86, and MS.DOS) and to take advantage of RS232 technology. MICROLAN was tested successfully for file transfer at up to 4800 baud, and suggestions have been included as to possible uses for MICROLAN in the military environment. Additionally, possible methods for upgrading MICROLAN are also included. Additional keywords: electronic mail, computer program documentation; theses.

TABLE OF CONTENTS

# LIST OF FIGURES

# I. INTRODUCTION

The age of automation has established its foothold within the civilian as well as military communities. Very few jobs are left unaffected by computerization. Military or civilian, the "boss" is only as competitive as his or her information - which must be accessed or acquired from outside sources. Application of this to current automation implies sharing database information between computers. For example, consider two users of a manual, each holding part of that manual in their files. If one of the users suddenly needs information from the other user's portion of the manual there needs to be a means for access. In certain applications, especially military, speed is essential in information transfer. Transcribing data in the manual mode is ineffective due to slow response time and an increased chance of error.

One option that will help eliminate some of these problems is networking. There are two different types of networks presently in existence, long-haul and local area. Long-haul pertains to large geographic areas; examples being TELENET, TYMNET, ARPANET, and the public telephone system. Local area concerns itself with a much more restricted geographic area; examples being ETHERNET, OMNINET, PCNET, WANGNET, and LOCALNET 20. Our interest lies within the field of local area networks (LAN).

One major distinction between types of LANs is baseband versus broadband. Baseband is limited to transmission of bits of information while broadband allows transmission of video, audio, and digital data. Normally, baseband is also limited to single channel connections while broadband allows multiple channels for transmission. For our interests, the key factor is that broadband requires expensive hardware and software. We have therefore focused on a baseband LAN system.

To focus our efforts even further, we compared asynchronous versus synchronous data transmission. In asynchronous transmission, machine interface is controlled by start and stop signals (handshaking) between the two microcomputers. Synchronous transmission requires both micros to operate on the exact same timing signals, either through a shared timing circuit or highly accurate timing systems at both ends. By themselves, no two computers - even of the same make and model - can be guaranteed to operate synchronously, and the cost of highly accurate timing is out of range for the small user. Since the military needs a low-cost, readily available system (see Chapter IV), we focused our efforts on asynchronous, baseband LANs.

LANs have become a common addition to many large organizations. They provide communication within a building or small groups of buildings, such as on a campus. Specific configurations depend on the volume and characteristics of the traffic, and the demands placed on the system. Local computer networks can also share peripherals. Sharing printers can be very cost-effective. By reducing the need of multiple printers, the idle time is kept to a minimum. Also, if one breaks down, the operator takes advantage of one of the other shared printers. Electronic mail may also justify the network depending on its implementation.

The military, which for our purposes is another large organization, has many of these same needs. The acquisition process varies between organizations. Standard acquisition methodology for the military is to evaluate present and future needs, come up with a list of requirements which a system could accomplish, competitively bid the system, and then await completion by a chosen manufacturer. This is an over-simplified description of the actual process, but it will suffice for this discussion.

A large factor in determining which systems will be actually be acquired by the military is availability of funds. To alleviate some of the monetary problems, DoD tries to incorporate systems which can be used by the four major services; Navy, Air Force, Army, and Marine Corps. Lead-time required to obtain an operable network satisfactory to one or more services is usually measured in years, and the final product usually neglects the needs of some echelon levels.

To fill the time-to-acquisition gap, we have developed a program, MICROLAN[1] for transfering computer files between the types of microcomputers that are already present in the field. Since we use equipment and technology already available in the field, our miniature LAN can be quickly and cheaply installed. Cost is discussed in Chapter II.

To simplify the design of MICROLAN and ensure the flexibility of operating on different microcomputers, we did not include protection from collision of data on the transmission medium if more than one micro tries to transmit a file at the same time. If a second user tries to transmit a file while a file transfer is in progress, the receiver for the original file transfer will have a checksum mismatch with the original sender. Eventually the file will make it through, but transmission will be disrupted until the second transmitting unit decides to stop sending. As a result of this problem, a second file transfer session cannot be safely started until the first file transfer has been completed. We refer to this limited capacity of only one file transfer on the net at any given time as "low density" traffic.

---

[1]Copyright 1985, Roger D. Jaskot and Harold W. Henry

MICROLAN allows transfer of computer files from a disk (hard or floppy) in one micro to a disk in another micro. The files can be man-readable data or text files or machine readable operation code. MICROLAN also takes advantage of common equipment that is inherent to the majority of microprocessors. As an example, the RS232 is a standard interface connection on most microcomputers. Any medium that accepts the RS232 (hardwire, AC modem, phone modem, fiber optics, etc.) can be used with MICROLAN, whereas other file transfer programs are company/device dependent. A particularly cheap medium would be the use of existing power system wiring (ie. - the wall power plug) as an access to other computers. However, a device called an alternating current (AC) modem would be necessary to make such a connection. Such a device is now available off-the-shelf.

The training required to use MICROLAN is minimal. The necessary computer skills should already be present for those personnel presently working with the military systems. Actually, only routine clerical skills are needed for proper operation.

# II. PROGRAM DESCRIPTION

The intention of this chapter is to give the reader a broad overview on the purpose and functions of MICROLAN. A detailed description involving assembly language is available in Appendix A.

MICROLAN is a file transfer program intended for use with most microcomputers. It's a very straightforward program designed to reduce the need for manual transcription and delivery of files. The reduction of error inherent with the manual transcription is a benefit with this system. The program can be used between microcomputers within an office or between different buildings. The design of the program is based on the lower networking levels (see Chapter 4). MICROLAN is readily available and can be implemented in any size military command or installation. It is presently written in assembly language for CP/M[2] - 80, CP/M - 86, and MS.DOS.[3] (Copies of each program can be found in Appendices C,D, and E respectfully. Very minor changes will have to be made to the program, depending on which microprocessor is used). Using one of these three versions as a basis, MICROLAN could be translated to operate in another language, if needed. However, we feel that these three versions should be compatible with the majority of the systems presently operating in the military. MICROLAN has been tested on Northstar, Apple, and IBM microcomputers.

In MICROLAN, we have improved on asynchronous (character-by-character) transmission by adding the higher speed of synchronous transmission. Rosner states that:

---

[2]Registered trademark of Digital Research

[3]Registered Trademark of Microsoft Corporation

Low-speed, asynchronous character-by-character terminals operate in typical speed ranges of 75 to 600 bits/second. This class of terminal is a nonintelligent device and thus cannot respond to the protocol features of a packet switch interface. High-speed, synchronous block-by-block terminals operate in typical speed ranges of 1200 to 9600 bits/second. This class of terminal can range from non-intelligent - which can only respond to a very limited set of level 2, link-control commands - to highly intelligent, processor-controlled terminals - which can support all packet switched network protocol features with the possible exception of multiple simultaneous logical connections. [Ref. 1: p. 118]

MICROLAN is asynchronous in transmission method. However, due to its structure, MICROLAN can operate at the speed of synchronous transmission (theoretically, as high as 19,200 baud). The value of 19.2 kbaud is the practical limit for RS232 hardware units.

We realize that there are systems with faster transfer rates presently available, however, the hardware required increases the cost of the system dramatically.

Since MICROLAN is a baseband LAN, we will compare its cost to other baseband LANs. All of the costs given assume that the user already has the microcomputer to be used in the system. The cost for Ethernet is $988 per user, with a minimum starting cost of $2202 [Ref. 2: p. 151] Omninet costs $650 per user with a $2230 minimum [Ref. 2: p. 141] and PCNet costs $742 per user with a $1762 minimum [Ref. 2: p. 129] MICROLAN has no minimum cost for software (the program listings are in Appendices C, D, and E) or for a starter kit. The cost for hardwire connections should be a maximum of $50 for a small system, and for an AC modem connection would be about $150 per user. By restricting MICROLAN's capabilities, we have been able to provide a readily available, low-cost LAN system. User interface with MICROLAN is minimal, and is driven by on screen instructions once the user has initiated program execution.

MICROLAN is intended for use by organizations as a convenience, and as an alternate means of sharing information that is not time-sensitive. By "convenience", we mean that you can get or send the information without having to leave your work station. "Time-sensitive" means that the information loses value with every extra second that it takes to get to the receiver.

MICROLAN requires action on both sending and receiving ends to initiate transfer. The users must meet on the net at either a standard time (e.g., 0900 each Tuesday), or they must coordinate just prior to starting file transfer (e.g., a phone call saying meet me on the net and send the file). Timing as far as whether the sender or receiver starts first is not critical; however, the send portion of MICROLAN dies after about a minute with no contact. If this occurs, the computer must be rebooted. The receive portion of MICROLAN will wait indefinitely for contact from the sending micro.

To show how MICROLAN is used for file transfer, consider Capt X, who needs information from Lt Q on a new project. (Procedures would be the same if Lt Q needed a printout of a file, but Capt X had the printer.) Assuming that the physical connections are already made, Capt X calls Lt Q and tells the Lt to send the file on the net in 15 minutes. At that time, both Capt X and Lt Q type "MICROLAN" followed by a carriage return. If they are using the CP/M - 86 or MS.DOS versions, the Capt and Lt will now be asked to select transfer baud rate from a menu (see BAUDMSG in Appendix F). The selected baud rates must match. Next, as the receiver, Capt X types an "R". The Capt is asked whether to write the file to the A, B, C(for CP/M-86 or MS.DOS), or default disk drive. Once the Capt selects the appropriate disk drive, his or her micro is in the receive mode and proceeds under control of MICROLAN until file transfer is completed. As the sender, Lt Q types an "S" to enter the send mode. The Lt is then directed to enter the name of the file to be transfered in the format "B:Filename.Filetype" where B represents the B disk drive. If the file was in the C drive, the Lt would replace the B with a C. If Lt Q typed in the filename in the format "Filename.Filetype", MICROLAN would assume that the file is on the default disk drive. Once Lt Q has entered the filename, MICROLAN takes over and no further action is required of either user unless Lt Q decides for some reason to abort file transfer. If the Lt should decide to do so, he or she could stop sending the file by pressing the <Control> and "C" keys at the same time just after a "*" has been printed on the screen to indicate that a 128-byte frame has been acknowledged.

MICROLAN begins by sending and receiving "handshaking" indicators allowing the micros to become synchronized. After the program is satisfied that they're in sync, the transmitting micro sends the filename and ensures through error checking that the correct filename was received. After this acknowledgement, the transmitting micro begins sending 128-byte blocks of information across the line. A checksum is calculated throughout transmission and is checked after each block is sent. If it checks good, then transmission is continued with the next block. If an error is detected, then the block is retransmitted. This procedure is continued until the entire file is sent. When the end of the file is reached, an "end-of-file" indicator is sent, telling the receiving micro that no further file information will be coming and to go ahead and close the file. A handshaking process then takes place, acknowledging file transfer is complete, and that both micros are ready to return to the operating system. Both micros then exit the program and are ready for the operators next desired command. It must be noted that, while MICROLAN is executing, the two microcomputers cannot be used to perform any other operations.

There are some safety factors incorporated for ease of operation. First, if the transmitting operator decides to abort transmission at any time, an input of "control C" will execute the abort. A message will let the receiving operator know that file transmission was aborted and that an empty file exists under that filename. Second, if the receiving file already has an existing file with the same filename, or if the transmitting micro cannot find the desired file, execution will stop, advising both sides of the situation. Third, if the receiving micro has a full disk and cannot receive the entire file, transfer will be aborted and both the operators will be advised. Appendix A expands on the above routines if any clarification is needed.

MICROLAN has been tested for operation in sending both man-readable text/data files and machine-language command files. It has been tested for transfer at 1200 baud between two Apple micros, two Northstar micros, Apple to Northstar and Northstar to Apple. Tests were also run at 4800 baud from Northstar to IBM PC , IBM PC to Northstar, and between two IBM PCs. Operation of MICROLAN was also tested at 9600 baud between two IBM PCs; however, the code logic used in MICROLAN proved unable to cope with the timing problems at this speed. Future revisions could overcome these problems. These tests were performed using hardwire connections; however, results should not vary with different connection media.

## III. MILITARY APPLICATIONS

MICROLAN can be used by the military to help meet current information sharing needs. Its attributes help alleviate certain problems that exist in current systems or that arise when obtaining a new system. One major problem the military encounters is the time delay that exists between statement of need and delivery to the service. Too often is the case that when the finished product is finally fully operational, the "threat" is at an advanced stage, thus making the system somewhat obsolete. Rather than waiting for a technological breakthrough to occur that will take care of any possible future threat, a system has to be deployed to counter the current "threat". MICROLAN is available for immediate implementation. It can be used by itself, as an enhancement to existing systems, or in the development of future systems.

The option exists for intra- as well as inter-service use. If the need for joint interoperability doesn't arise in a specific situation, MICROLAN is still fully operational within the realm of a single service. Inter-service use poses no major changes either. The same existing hardware and software are still used. MICROLAN can be quickly adapted to almost any microcomputer, thus overcoming the profusion of dissimilar equipment in the field.

Required operational training of personnel is kept to a minimum with MICROLAN. The military employs a vast range of users, varying in educational backgrounds. MICROLAN's ease of operation is limited only by the most rudimentary knowledge of the typewriter keyboard.

Cost overruns, scheduling delays, contract disputes, and a myriad of other pitfalls plague the Department of Defense budget. MICROLAN is inexpensive, available, and easy to incorporate. Adhering to these attributes, MICROLAN would not be a financial burden to the military. "Word of mouth" is one of the best promoters of a new product. Enhanced by promotional meetings, computer bulletin boards, satisfied users, etc., MICROLAN's usefulness will hopefully be widely disseminated to all facets of the military.

This chapter presents some examples of how MICROLAN could be used by the military. It will also cite some examples of how the military is presently trying to automate information distribution.

## A. NAVY

The U.S. Navy has undergone a major face-lift over the past two decades. Significant breakthroughs in technology has offered tremendous advances in ship-building design and associated weapon systems. Due to these advancements, decision-making by the warfare commander has been quickened by shorter planning cycles, dissemination of orders, and resulting outcomes of those orders. The "real-time" response to any attack has had to be critically shortened in order for present day operations to be successful. Turn-around time in paperwork has also gone through many changes in attempts to minimize slack time caused by tedious, but necessary, record keeping. Personal files, parts orders, and safety statistics are just a few of the necessary information requirements for any large organization.

Whether it be in an operational setting, such as the Combat Information Center (CIC) aboard a ship, or in a shore-based supply facility, the Navy is always looking for ways of reducing the workload placed on its personnel. One such system that the Navy is presently pursuing is the ZOG

[Ref. 3] system, which has been placed on board the aircraft carrier, the USS Carl Vinson, as one of three microcomputer networks. This example will provide an idea as to what the Navy is looking for in the field of computers.

ZOG is a general-purpose human-computer interface system that combines the features of a database system, a word processing system, and an operating system shell. This system is a distributed database system implemented on a network of 28 high-powered personal computers (PERQS), interconnected via a wideband local area network (Ethernet).

The uses of a local area network with computers are seemingly endless. A few examples of the ZOG system will suffice. On board the USS Carl Vinson, ZOG has been used as a software management database, well suited for structured software development. It has also been extensively used to implement forms of electronic communication, such as electronic mail, bulletin boards, and teleconferencing. In a more advanced area, ZOG was used for project management; to develop multi-level task structures which could be used not only for planning, but for implementing and evaluating as well. Other areas that were explored were training, interfacing with an existing system, and retrieval of emergency operating instructions (in this case, for commercial nuclear power plants). As with almost any new system, there's always room for improvement. An extension of ZOG is the Knowledge Management System (KMS). In KMS the model of a frame has been extended to include graphical as well as textual items.

The ZOG example provided a good insight as to where the Navy is looking in terms of newer technologies. Akscyn and McCracken brought out a good point in their report (Ref. 2). That is, how the users of the system can make their work usable by others, especially since there are few situations in the real world where people do not depend on interaction with others to accomplish their work.

Our file transfer program, integrated in a local area network, could alleviate some of the problems. To gain a better perspective on the usefulness of this program, let us state that this project was not intended to be used in a time-sensitive environment. An example of that would be in use with the Navy Tactical Data System (NTDS) updating friendly as well as enemy positions. In this situation, seconds are critical concerning command decisions.

One area where this system could be very useful is in the supply system. It is irrelevant as to whether the supply department involved is shore-based or afloat. Transferring files, part orders, etc., between buildings or ship compartments would drastically reduce the manual labor presently involved. Consolidating the payroll system would greatly reduce the space required for all of the necessary paperwork.

Electronic mail would be a good use also. The administration departments would find it useful in preparing command-wide bulletins (e.g. Plan-of the Day) or collating fitness reports. The communications department could utilize the system for drafting message traffic. Instead of congesting the commanding officer's desk with messages awaiting approval, they could be sent to his disk, which he could then address at his own leisure, returning finished copies at will. The maintenance department could "converse" with the supply department in a more organized manner concerning needed equipment. The safety department, in conjunction with the maintenance department, would be able to pass or collect necessary statistics needed for periodic reports.

These are just a few examples which could be incorporated within a command. They would not have to utilize these opportunities all the time, however the option would be there. The main benefit of this system is elimination of transferal of paperwork between departments (or even within departments). Having a condensed file of needed information on one disk would definitely reduce the amount of lost information due to scattered, and inadvertently discarded, paperwork. One important aspect to keep in mind is that the manual method of information transferal would still be available, if needed for one reason or another.

## B. ARMY

The U.S. Army does not enjoy the luxury of being numerically superior to present day opposing forces. Even though the Army has a slight qualitative and technological advantage, the threat combines its numerical advantage with its increasing weapon and combat technologies to at least nullify the slim margin the U.S. presently holds.

The Army, like the other services, tries to utilize as much new technology as possible to sustain this margin. There is more information on and about the battlefield today than ever before, however, the staff essentially still processes the information in a manual mode. There are some automated procedures, but the bulk of the system contains mostly manual procedures.

In order to alleviate some of these problems, the Army has introduced CPASS (Command Post Automated Staff Support System). [Ref. 4] The primary purpose of the CPASS system is to provide automated assistance in performing staff functions. The automation devices and software of the system are tools that expand the staff's capability to handle more information and to utilize the information more efficiently. Some of the intended uses for CPASS are:

a) An information processing system to develop and execute staff plans and operational orders.

b) Provide a near-term staff wide command post automated information distribution and decision support capability.

c) Provide more real time and near real-time accurate information to commanders and their staffs.

d) A graphical situation display and hard copy overlay capability.

e) Automation for both tactical and garrison applications without a requirement for intensive train-up or transition to meet deployment or operational requirements. Such a system is required for daily use, not just to support the deployed command post.

f) A initial capability for the evolutionary development of concepts, doctrine, procedures, hardware, and software for the continuing automation of command post staff activities.

g) Capability to support the dispersed command post in accordance with current doctrine. It must demonstrate the additional operational and organization changes required to support the dispersed command post.

Items b,e,f, and g are along the same intentions (automation of commands, information sharing, and minimal training requirement) as that of MICROLAN. The hardware/software make-up of the CPASS is unquestionably larger and more complex than our system. However, some of their components and structures are used for the same basic purposes as ours. A few of them are:

a) Within the command post cluster, devices are interconnected by a local area network (LAN). The LAN is physically versatile and can interconnect devices in all shelter configurations of a command post cluster (expandable shelters, buildings, or within an armored command post vehicle or van). Media types to be used in LAN include twisted pair wire, coax cable, or fiber optics.

b) A Network Computer Unit (NCU) containing a network interface element and a microprocessor performs data routing functions within the work station, and between the work station and other cluster devices.

c) The file storage device stores elements of the data bases of other command post clusters in anticipation of combat loss or equipment malfunction.

d) The communications processor, in conjunction with the communications devices of the area communications system, provide end-to-end message transport service to allow essential interstation communication, such as message routing, data base interactions and graphics data transfer.

The required personnel training for CPASS is very similar to ours. The introduction of CPASS is not projected to require increased command post manning levels, new military occupational specialties, or Army skill indicators. Operators are those who are already assigned to command post staff functions. Routine clerical skills are the minimum essential personnel qualification skills needed to operate the system (i.e., typewriter keyboard, filing, etc.).

The CPASS example is a good indicator of what direction the Army is heading in terms of computer use. Our system contains many of the same qualities that the Army desires and requires.

## C. AIR FORCE

The Air Force is deeply involved in networking headquarters and tactical functions. However, the focus is on expensive broadband networks and tends to neglect the smaller users. The report of the Hardened Tactical Air Control Center (HTACC) even states that:

> Of the three major functions in the HTACC--intelligence, operations, and logistics--the direct automation support from the CONSTANT WATCH Program is largely restricted to the intelligence and operations functions under current plans. Indirect support for logistics functions will result from the automation of intelligence and operations functions which logistics uses and from secondary use of the communication capabilities implemented under the CONSTANT WATCH program. Eventually, logistics automation requirements must be addressed, and, hopefully, integrated with the intelligence and operations activities. [Ref. 5: p. II.4]

The HTACC involves use of high-cost broadband technology. Our method could use the power cables that must be present anyway, and requires only an AC modem and minimal cabling in addition to the RS232 that is standard on most microcomputers. Since the logistics requirements are not real-time, and shouldn't be extremely high density in traffic, they could be supported by MICROLAN. Reports on status of supplies and requests for movement of supplies are the types of traffic that could be expected on the system. Also, minor information transfer between control positions in the TACC could be accomplished on our system, reducing the workload on the real-time LAN system.

The Air force is also working on a LAN (PENTANET) for the Pentagon to provide:

a) The exchange of data between local and remote interactive Keyboard Video Display (KVD) terminals and local and remote processors, wherein local and remote connote devices within and exterior to the Pentagon
b) The electronic exchange of variably formatted reports and documents between local and remote workstations
c) The local and remote distribution of digitally encoded graphic and facsimile products
d) The transfer of data files between local and remote processors and between local and remote peripheral devices

e) The transfer and distribution of teleconferencing and commercial video and associated analog voice, and low speed analog and digital control signals

f) The switching and exchange of analog and digitized voice signals between users. [Ref. 6: p. 9]

Here, our program could be used to supplement the PENTANET as an interoffice message transfer system, reducing the major net's workload. The memos could include coordination on letters or short documents that can also be sent using our program. File transfer via MICROLAN is not limited to text. If one office has a program that another would like to use, it can be passed over MICROLAN, even if the computers are not the same brand name product.

Another Air Force function that could make use of our program is the Base Information Transfer System (BITS). BITS is the base mail system. Memos, completed forms, blank forms, appointment notifications, general mail, and coordination copies of documents are transported around base using this system. Base administrative personnel pick up the correspondence, take it to a central processing office, and then deliver it to the destination. From past experience, BITS has been known to lose messages, and the only way that has been recommended for improving timeliness of service is an increased number of delivery runs [Ref. 7]. To implement more runs would require more personnel, therefore increasing costs. Use of MICROLAN would have a low one-time cost and almost no upkeep. One additional requirement for implementing our system on a base-wide basis, if power system wiring is to be used as the net, would be installation of capacitors on power transformers to allow the LAN to cover a wider area of the base. The capacitors would allow our signal to pass through the transformer while preventing the AC power from crossing over. Installation would be only a minor problem. Memos and coordination would require no modification to be passed using MICROLAN. Reports and supply requests could be formatted and transferred to action agencies for printout on the receiving end.

One possible use was suggested by Hq Tactical Air Command, Tactical Air Forces Interoperability Group (TAFIG). TAFIG identified a need for transfering wing or squadron databases to computers onboard aircraft. This would require either a disk system in the aircraft or storage of the program on a programmable memory chip, which would be more reasonable. This system would provide pilots with data through the onboard computer, decreasing some of the time that would be required for briefings on the ground.

## D. MARINE CORPS

We contacted the Marine Corps Command and Control Systems Office at Camp Pendleton, California to determine what types of network systems they were looking for. They indicated that they have immediate needs for interoffice file transfer and mailgram systems, both of which MICROLAN can provide. They also have a need for a tactical message transfer system in the 9600 baud transfer range, which would have to be able to be sent via encryption or other secure means. Our system of transfer using RS232 technology and the buffers built into MICROLAN should allow transmission via a variety of media, including fiber optics. Although we have not tested MICROLAN with encryption, we do not expect serious problems in doing so. In addition to these uses, the CPASS system mentioned under the Army section of this chapter contains uses that would also apply to Marine operations.

## E. CHAPTER SUMMARY

In this chapter, we have presented several possible uses for our file transfer program in filling present requirements of the four service branches. We have not attempted to enumerate every possible application of our program, only some representative uses for each service. There are most certainly more file transfer uses that exist that MICROLAN can be applied to. The key prerequisites for using our system are that the data is not time sensitive and that traffic is low-density. A review of our suggested uses for MICROLAN shows that there are applications throughout the spectrum of service organizations-whether in the back office or on the battlefield, shipboard or aboard aircraft-that meet these prerequisites.

Use of MICROLAN for interoffice memos could be applied to any installation or organization. For example, the Navy Postgraduate School has a need for an inter- and intra-departmental mailgram system. Intra-departmental networking should be no problem for an AC modem system, since members of a given department are usually grouped together in the same building. For inter-departmental use or departments that are spread across campus, capacitors would have to be used as mentioned in the Air Force section of this chapter. If use of the system becomes saturated, methods identified in the Conclusion for separating nets could be employed.

It is important to note that the use of RS232 interface technology allows a varied means of connection between sending and receiving units. This is a significant factor in MICROLAN's flexibility. Another aspect aspect of MICROLAN that contributes to its flexibility and interoperability is that it is confined to the lower levels of the International Standards Organization (ISO) model. This ensures that neither higher levels of computing power nor specialized components or exotic software are required to implement the MICROLAN system. Chapter 4 explains how MICROLAN fits into the ISO model.

# IV. MICROLAN AND THE NETWORK MODEL

To understand where MICROLAN fits into the International Standards Organization (ISO) Open Systems Interconnection (OSI) model, the basic basic idea of that model's concepts are given. The ISO OSI model consists of seven layers (levels) corresponding to computer functions and interconnection. These range from a basic physical layer to user interface. Figure 4.1 is the standard representation of these layers.

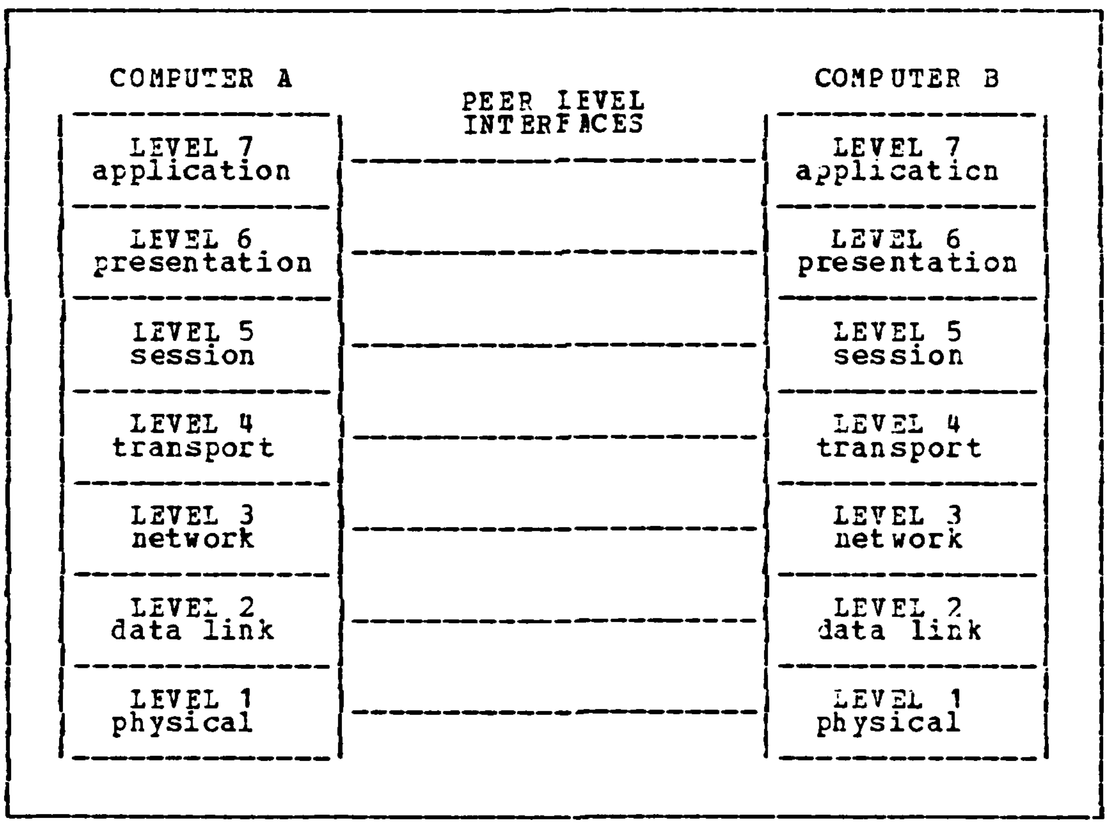



Figure 4.1 International Standards Organization Protocol Model.

## A. MODEL OVERVIEW

The seven layers of the OSI model are discussed in short and at length by many authors on networking computers. The best summary that we found was Roy Rosner's book in which he states:

> The lowest level of the ISO protocol hierarchy is the physical level, where previously defined standards were applied to define the physical interface. By physical interface to the network we refer to the pin connections, electrical voltage levels, and signal formats. Level 2, known as the data-link level, controls the data link between the user and the network. This level defines data format, error control and recovery procedures, data transparency, and implementation of certain command sequences. For nonswitched networks, or the interface of simple terminals with computers through point-to-point services, generally only levels 1 and 2 are required. Networks designed by a single manufacturer around a single product line, generally do so with a combination of level 1 and level 2 protocols.
>
> Level 3, the network level, defines most of the protocol-driven functions of the packet network interface, or the internal network. It is at this level that the flow-control procedures are employed and where switched services are initiated through a data call establishment procedure.
>
> Level 4, known as the transport level, assures the end-to-end flow of complete messages. If the network requires that messages be broken down into segments or packets at the interface, the transport level assures that the message segmentation takes place and that the message is properly delivered.
>
> Level 5, the session control level, controls the interaction of user software, which is exchanging data at each end of the network. Session control includes such things as network log-on, user authentication, and the allocation of ADP resources within user equipment. Level 6, the presentation level, controls display formats, data code conversion, and information going to and from peripheral storage devices. Level 7, the user process or user application level, deals directly with the software application programs that interact through the network.
>
> Although at levels 5, 6, and 7 the protocol is defined from a functional viewpoint, implementation of standard software that can operate at these levels has been slow. The software at all of these levels (often referred to as peer-level software) tends to be both equipment and application dependent. However, the layered approach to protocol development achieves a degree of isolation and modularity between the various layers, so that changes in one level can be made without changes in any other level. [Ref. 1: p. 109]

MICROLAN's structure fits into the lower levels of the OSI model. For our purposes, it is important to note that layer 1 signaling modes include: full duplex, half-duplex, synchronous, asynchronous, balanced, etc. There are also several standards that exist at layer 1. For example, there is EIA's RS232 and RS449, and CCITT's X.21, V.24, and V.35. [Ref. 6: p. 97] How MICROLAN functions in each layer is explained in the following pages.

## B. PHYSICAL LAYER

For the Layer 1 interface, we take advantage of RS232 technology, thus providing a standardized physical interface for MICROLAN. This eliminates the problem of matching high and low voltages for different computers. Normally each individual bit is regarded as an entity for Physical Layer purposes; however, in our design, an 8-bit byte is used as an entity for transmission of data. This is the smallest segment of information handled by a microcomputer's accumulator, and is the ASCII representation of data characters. It is in this layer where we get our greatest flexibility. This flexibility arises from the fact that a variety of methods exists for linking one RS232 to another, as mentioned in Chapter I, providing the user with options in the type of hardware they can use.

## C. DATA LINK LAYER

In this layer, our data is grouped into a 'frame' of 128 bytes. This number equals the storage capacity of the Direct Memory Access (DMA) buffer that is standard on microcomputers. A file is broken into frames and reassembled using the microcomputer's operating system commands. On the sending side, a read sequential command breaks out the sequential frames by reading 128-byte blocks into the DMA for transmission. On the receiving end, the DMA is filled by the 128 bytes that were sent, then a write sequential command places the frame into the new file in sequence. By doing this, we prevent having to develop software methods for sequencing frames.

MICROLAN performs error checking on a frame to frame basis. Within each frame of data, a checksum is calculated by both sender and receiver and compared at the receiving end. If the two checksums don't match, the receiving micro informs the sending micro, which then retransmits the same frame of data, repeating until the frame is acknowledged as received correct or the sending user decides to abort file transfer. Combined with the error checking, we built buffers in to allow slower micros (e.g., Apple versus Northstar) to catch up to their faster counterparts.

Since we use the DMA regulated 128-byte block for our data frame on both ends of transmission, the amount of data sent at one time can never exceed the receiving micro's buffer capacity. Therefore, MICROLAN doesn't require a 'buffer space left' notification that would normally occur in this layer. [Ref. 8: p. 17]. Instead, it is at this level where the receiving micro checks for free disk space and informs the sending micro to abort file transfer if there is no more disk space for storage. Finally, one frame must be acknowledged as received and correct before MICROLAN will send the next frame. This eliminates the problem of duplicate or lost data frames.

As stated by Rosner in the above quote, this is the highest level required of simple, nonswitched networks. However, in order to allow the user some control of MICROLAN, we do provide some features in the Session layer.

## D. SESSION LAYER

This is the final layer used in MICROLAN. Here, the user invokes MICROLAN and initiates connection by selecting send or receive functions. During this process, the user also selects which disk drive (default or an alternate) for accessing or storing the file. The sending user's option of aborting file transfer also falls under the definition of this layer. See Chapter II for operation instructions.

## E. SUMMARY

The majority of MICROLAN's activities occur in the lower two layers of the ISO OSI model, as seen above. As a result, user friendliness is limited. Also, as mentioned in Chapter II, MICROLAN monopolizes the computer, allowing no other operations. MICROLAN is being used as the basis for a higher level network system by a fellow NPS student, LCDR Jeanie Egbert, in her thesis, "A MICROCOMPUTER NETWORK: INVESTIGATION AND IMPLEMENTATION". The combination of MICROLAN and her thesis provides a more presentation oriented structure. LCDR Egbert's LAN system allows the user to perform other operations on their micros while files are being transfered.

Since MICROLAN performs no Network Layer functions, no collision detection or prevention is provided - as mentioned in Chapter I. Flow control is limited to the link between one sending and one receiving micro on the network.

By limiting MICROLAN's main functions to the lower layers of the ISO OSI model, we have provided a simple, nonswitched file transfer system (LAN). MICROLAN is designed to operate on a variety of microcomputers, not just one product line. Therefore, we have gone one step farther than Rosner indicated in the first paragraph of his model description.

Further improvements can always be made to a program. We identify some of these possibilities in the next chapter.

# V. CONCLUSIONS AND RECOMMENDATIONS

## A. CONCLUSIONS

The need exists for a public domain low-cost file transfer system to provide an alternative to commercial systems (e.g., ETHERNET). The type of transfer considered here is low-density traffic that is not time-sensitive. Types of files include computer programs (both in text and in machine code), data, messages, and text files.

Our solution was to take advantage of standard RS232 technology that is used by all microcomputers. This makes MICROLAN capable of operating on a wide variety of micros. Writing MICROLAN in CP/M, CP/M-86, and MS-DOS versions of assembly language makes it compatible with-at a minimum-Northstar, Apple, and IBM-PC compatibles.

There are two key restrictions encountered when using MICROLAN for file transfer. First, MICROLAN monopolizes both sending and receiving micros so that they are not available for other purposes until file transfer is completed. LCDR Egbert's thesis, mentioned in Chapter IV, addresses this problem. Second, only one file transfer can be conducted on a particular net at a given time. This is because MICROLAN has no detection or prevention of on-line data collisions. Providing multiple paths in a given network area would reduce chance of collision and allow more than one transfer at a given time. A method for doing this is discussed in the second part of this chapter.

Our intentions were to keep MICROLAN simple, so that an in depth knowledge of computers is not required to use it. Once the RS232 connections are made (standard plug connectors make this step relatively simple), entering 'MICROLAN' from the console presents the user with easy to follow instructions including sample inputs that lead to file transfer. During execution, MICROLAN prints phrases on the user's screen, informing them of transfer status. These on-screen comments are shown in Appendix F. Also, a '*' is printed on the screens of both sender and receiver for each 128-byte block sent. A 'b' is printed on the sender's screen for each unmatched checksum. This feedback allows the users to see that transfer is occuring and determine if problems exist (e.g., several b's in a row). The sending user can exercise the option to abort transfer if the bad checksums persist.

Two safety features are included in the SLAVE subprogram. If the receiving disk already has a file with the same name as that of the file being transfered, file transfer is aborted and the users are informed of the problem. This protects an existing file from being overwritten by a new file. The other safety feature is activated when the receiving disk runs out of memory space. When this occurs, file transfer is aborted and both users are informed. The file is not closed under this method, so file transfer is 'all or none'. We chose this method because, especially in the case of command files, serious problems could arise from partial transfer of files. Command files cannot be readily repaired by the receiver by merely 'typing in' the missing parts.

In Chapter III, we discussed various military uses for MICROLAN, based on a search of network requirements from the four services. The main criteria for using MICROLAN is that the information requirement must not be time sensitive and traffic must be low-density. MICROLAN would also allow organizations that can't justify the expense of a high-powered network, an option for a low-power, medium speed (up to 19,200 baud) network at a much lower cost. Use of RS232 standards means that the user can take advantage of whatever connection medium is readily available, whether it be telephone, power lines, or direct wire. This would also help lower costs.

MICROLAN operates mainly in Layers 1 and 2 of the ISO OSI network model, as explained in Chapter IV. It is a simple, transfer-oriented system. User interface at the upper levels is the minimum necessary to operate the program. This was done deliberately to maintain maximum flexibility in rewriting MICROLAN to run on different micros.

MICROLAN has been successfully tested for operation using hardwire connections between the RS232s, at rates up to 4800 baud. When used with interrupt driven programs such as LCDR Egbert's, where timing problems will not exist, we expect transfer speeds of 19,200 baud to be possible. Replacing the hardwire connection with modems, fiber optics, or any other type of medium should not affect operation of MICROLAN.

In MICROLAN, we have provided a flexible, low-cost mini LAN as an new option for information transfer. Of course, improvements can always be made to any program, so the next section presents some that we recommend for MICROLAN.

## B. RECOMMENDATIONS

As mentioned earlier, MICROLAN does not provide collision detection and prevention. One project for further research would be to develop program code to incorporate collision detection and avoidance into our program.

One change that would only require minor modifications is to return the user to the Send/Receive/Exit menu after file transfer is completed. We felt that returning the user to the operating system was more appropriate, but others may feel differently. At this same level-i.e., the menu-it would also be possible to add the ability to send more than one file in a session. This would require changes in the File Control Block load subroutine, as well as a change in the end-of-file subroutine to loop back to send the next file.

It is possible that noisy transmission lines could cause problems with MICROLAN's checksum procedure. A subject for further research would be development of an algorithm for noisy line error checking perhaps by using cyclic redundancy.

As presently written, MICROLAN dumps files only to a disk system. To add flexibility, menu driven subroutines could be added to allow file transfer directly to other peripherals. This would allow one user to 'borrow' another's printer without moving it. Of course, file transfer would be slowed by the limited speed of the printer.

Use of MICROLAN as a Bulletin Board system would require a menu item in addition to Send/Receive/Exit. Subroutines to execute this option would have to use the Console Buffer and Random Access Memory of the micro to store bulletin items. The first bulletins would print on the user's screen, with following items stored in memory. The system would have to allow the user to page through the bulletin items using console keys. The option to send as well as receive items while retaining the previous items would also be helpful.

To allow up to 500 micros to communicate in a given area, the net can be broken into separate subnets. Each subnet would operate on a different frequency channel as set up by a central controller. In the example of the NPS net requirement, one channel could be for the Superintendent, one for logistics, one for each of the departments, etc. In a one megahertz band, there could be 10-20 channels depending on baud rate. Since this is hardware driven, no software change would be necessary. However, channel selection could conceivably be software driven. If users are on several nets, they could use scanners to 'listen' for messages on the different nets in the same manner as radio scanners are used to listen for messages on Citizen's Band frequencies.

Tied in with 'listening' for messages, subroutines could be added to allow each user to have a personal identification number (PIN), assigned by net control. The micro would listen for messages to all users or to their PIN specifically, ignoring all others. This would operate best in conjunction with higher level programming, such as LCDR Egbert's program, that would allow the user to perform other computer operations while MICRCIAN is looking for messages.

Our final recommendation is one that would make MICROLAN operate as a token ring network. On board ship, where power is not a problem, the micros could be left on continuously (actually this is better for the micro). In conjunction with the PIN idea, software changes would have to be developed that would allow MICROLAN to be used as an intercom system. One user would control the intercom, passing control to other users as they have the need to ask or answer questions. Control of the intercom would then be passed back to the master user.

Obviously, we have not covered every possible use or improvement for MICRCIAN, but we hope that our description of MICROLAN and its possible uses has planted a seed for future research and expansion of low cost LANs.

## APPENDIX A

### DETAILED PROGRAM DESCRIPTION

Prior to writing assembly language code for MICROLAN, we developed a flow diagram to show what we wanted to accomplish with the program. We developed the Master and Slave portions in parallel, showing rendezvous points with connecting lines. This flow diagram is shown in Appendix B. From this flow diagram, we developed subroutines to actually execute the steps and loops required to transfer a file. The programs shown in Appendices C, D, and E include additions that make MICROLAN more user friendly (e.g., ability to select which disk drive or to abort transfer).

The MICROLAN file transfer program consists of two subprograms that operate on separate micros, with frequent rendezvous to ensure parallel operation. We used modular programming style and developed the Master and Slave subprograms in parallel to insure that the two would rendezvous at matching subroutines. Data transfer is up to 8 bits per byte (ASCII or standard hex). Buffers had to be added in the Master rendezvous subroutines to allow the Slave subprogram to catch up when using different micros.

Our program is written to be used on microcomputers using either CPM, CPM86, or MS.DOS computer program/manager operating system. To allow use on other types of systems, changes will be required in the assembly language code to match that used by the micro to be used.

The remainder of this description refers to the CP/M-80 version of MICROLAN except as noted. First, MICROLAN must know which language format will be used during operation. Language format refers to the type of commands inherent to the microcomputers system. For example, the Apple loads data from the input buffer into a memory address before reading it to the accumulator, while the Northstar takes data directly from an input port to the accumulator.

If the micro operates like a Northstar, the main change needed is in the definitions at the end of the program code. The user must change DATA EQU 04H to the number of the micro's input port and STATUS EQU 05H to the number of the status port. If the micro operates like an Apple, DATA1 and STATUS1 must be changed to reflect the micro's correct port numbers. The user should also verify that TXRDY and RXRDY reflect the correct values for their micro. Since the program is matched to Apple, Northstar, and IBM (and compatibles) types of microcomputers, MICROLAN should be useable by a wide variety of systems. There are seven subroutines affected by changing micros. They are POUT, STATIN1, STATIN2 and PIN. Slave also has the subroutine PIN1 that is affected, and Master has STOPS and GOCPM that are affected. They're easily spotted because they are the only subroutines that use the IF statement. Also, the appropriate constants will have to be added at the end of the program.

At the beginning of the program (see Appx C), the name of the micro that you are using must be set equal to TRUE. For example, APPLE EQU TRUE. The name(s) of other micros must be set to NOT TRUE. For example, NORTHSTAR EQU NOT TRUE. This activates the appropriate portion of the IF-THEN statements.

You will notice that we set the origin of MICROLAN at 0100 Hexadecimal(Hex). This is the standard position to load a program for execution. We then move the Stack Pointer to a higher memory address to prevent it from being overwritten.

To invoke MICROLAN, type 'MICROLAN' and press return. At this point in the program, if the user is using CP/M-86 or MS.DOS, they are asked to select transfer baud rate from a menu. For the CP/M-80 version, and continuing for the other two, the next step is the INIT subroutine asks you if you wish to send or receive. The HOLDING subroutine looks for a keyboard input until either an 'S', an 'R', or an 'X' is found. An 'S' sends the program to the MASTER subprogram. An 'R' sends it to the SLAVE subprogram. An 'X' returns the user to the main operating system of the micro.

MASTER first asks for the name of the file to be sent. The user can also identify at this point which disk drive to retrieve the file from. Possible selections are A, B, (for CP/M -86 and MS.DOS systems, the option for a C disk is also included) or default. The user specifies the drive by typing in the format 'B:filename.filetype'. If no drive is specified, the default is assumed. While the user is entering the filename, the FIILUP subroutines prepare the File Control Block (FCB) for receiving drive information and the filename. The FCB is located starting at memory location 005C Hex and is 32 memory locations long. It is the default filename location for all microcomputers.

HOLD1, FLUP, DONTFIX,FIXIT, and DSKSEL work together to read the disk drive selection, filename, and filetype and load them into the FCB.

Assuming that the proper wire connections have been made, the next step in Master is to send an 'R' on line to get the receiving micro's attention. Then the sending micro listens for a reply from the receiving micro. This is repeated until the sending micro receives an 'r' in reply. Master then prints a string to the screen to tell the user that connection has been made.

To ensure synchronization prior to sending the FCB, Master sends a Transmit Symbol (TXSYM). We use the ASCII equivalent for a DC2 control code as our TXSYM, chosen based on our determination that DC2 is not used frequently otherwise. Master then listens for a reply. As a buffer, this is repeated until the sending micro receives a 't' in reply. Before sending the FCB, an open file subroutine is called to insure that the file exists. If the file exists, the program continues. Otherwise, the session is aborted through a 'FNFOUND' subroutine. A 'QUIT' symbol, the ASCII Code for a DC4 control code, is sent online to tell the receiving micro that no file transfer will occur. Then a string is printed to the screen telling the user that no file was found and the program returns to CPM.

We use the B register to store the current checksum code, initializing it to zero (0) for reference. The HL register pair holds the address of the current memory (M) location for purposes of data manipulation. To send the FCB, we set the pointer in the HL pair at the starting memory location for the FCB (0C5C Hex). The next loop uses the current memory byte to perform the checksum operation and sends that byte on line until the current memory location holds a '0'. Once that '0' is sent on line, the loop is done, as the '0' denotes the end of that filename. The checksum code is a result of 'exclusive oring' the current data byte with the previous checksum code. The resulting checksum code is stored back in the B register. Use of a checksum ensures accurate data transmission.

After the end of filename has been sent, the sending micro waits for an 'r' indicating that the receiving micro received the end of file '0' signal. The checksum is then sent online. We save the checksum for possible retransmission, then clear the accumulator before listening for acknowledgement. If a 'b' is received, the checksums didn't match, so the FCB is resent using RSNDFCB. First, the checksum is recalled from the stack and moved to the accumulator. Then we offset the checksum by adding three (3) for use in synchronizing the two micros, and send the result online. Next is an indefinite wait loop that is left only when the reply matches RXACK ('r'). Following is a similar loop listening for a TXACK ('t'). When synchronization is set, the program jumps back to the subroutine TXFCB1 and resends the FCB. If a 'g' is received in reply, the transmitting micro proceeds to a wait loop for the receiving micro to catch up.

In the wait loop, the program checks for an input as many as 2000 times. If no input is received, the user is returned to CPM. When an input is received, it is compared to 'QUIT', a DC4 in ASCII Code. If a match is made, it means that the receiving disk already has a file of the same name and the program jumps to 'GOCPM1'. Here, a string is printed to the screen telling the user that the receiver already has a file of the same name and the user is returned to CPM. If the reply wasn't a 'QUIT', it is compared to a 'GOON' or continue symbol. If the input matches neither of the two, the wait loop is repeated; otherwise, a string is printed to the user screen that the file is being transmitted. Next, the program calls a read sequential subroutine to get the first(next) 128-byte block of data.

Prior to sending each 128-byte block, a 'CHECK' subroutine is called see if the sending micro is ready to transmit. 'CHECK' holds the program until the micro is 'transmit ready'. Then, for synchronization, a TXSYM is sent online. A listen loop follows, where the program checks for a TXACK or a disk full symbol (DSKFUL), which is a 'd'. If TXACK is received, data can be sent. If DSKFUL is received, it means that the receiving micro has no more disk storage space and a full disk (FULDISK) abort subroutine is called. The FULDISK subroutine sends a DONE symbol, a 'Z', online to acknowledge the 'DSKFUL' symbol. Then a string is printed to the screen telling the user that the receiver's disk is full, and the subroutine 'GOCPM' is called. First, a '0' is sent online to clear the output buffer of the sending micro and the input buffer of the receiving micro. Then the program returns to CPM. We found it necessary to send the '0' in order to prevent premature synchronization by the Slave micro. When we allowed the micro to return to CPM without this step, the Slave micro acted on whatever was left in the Master output buffer. This synchronization sequence is repeated until a match is made on one of the two expected inputs.

To separate the 128-byte frame from our control commands, MASTER now sends a Real Data (RLDTA) symbol, OCB Hex, to the receiving micro. MASTER then listens for an echo from SLAVE before continuing with file transmission. Again, this was necessary for synchronization between different types of computers.

When an echo is received, we set the H,L register pair pointer to the first location in the Direct Memory Address (DMA) buffer, which is 80 Hex. The DMA is 80 Hex, or 128 bytes of memory, and is the default storage location for data read to or from files by CPM. The checksum in register B is reset to 0 for each 128-byte block. Now a checksum is performed in the same manner as it was for the FCB, and the current byte is moved to the accumulator to be sent online. Then the H,L pointer is moved to the address of the next data byte in the DMA. This is repeated until the 128th byte is sent and the H,L pointer is incremented to 100Hex. When the last byte of the data block has been sent, the checksum is moved to the accumulator and sent online.

Next, we have another listen loop to allow the receiving micro to catch up. The program checks for input until one is received. Once an input is received, it is compared against the 'Bad' and 'Good' symbols. If it is 'Bad', the program jumps to a 'RESEND' subroutine. In 'RESEND', a 'b' is printed to the user's CRT telling them that the block checksum was bad and that same 128-byte block is to be resent. Then the block is sent again. If it is a 'Good', the program jumps to a subroutine to send the next 128-byte block, 'RDSQRPT'. Here, a '*' is printed to the user's CRT telling them that the block was successfully sent. Then the program jumps to 'RDSEQ' to read the next block of data to be sent. If there is no more data in the file, a TXSYM is sent online. The program then listens for a TXACK until one is received. Then 'EOFIL1' is called. First, a QUIT symbol is sent online. Then the program listens for an echoing QUIT symbol, repeating until the echo is received. Then a string is printed to the screen telling the user the file transfer is complete and 'CLOSIT' is called. A DONE symbol is sent online, the the transmitting micro listens for an echoing DONE in reply. Once the DONE echo is received, the program returns to CPM after sending a '0' online to clear the buffers. The listen sequence is repeated up to 26 times. If no match is made on 'Bad' or 'Good', we assume a problem and send the same 128-byte block again using the same procedures.

The 'POUT1' subroutine includes the ability for the sending user to abort file transfer. At any time during the program the user can enter a 'Control C' (ctrl C) from the keyboard to abort. 'POUT1' looks for this input every time it is called and, if 'ctrl C' is found, jumps to a 'STOPS' subroutine. The subroutine sends a CTRLC symbol online, then clears the accumulator and listens for a CTRLC echo from the receiving micro. This is repeated until an echo is received. The output buffer is then cleared and the program returns the user to CPM.

There is one subroutine, 'OUTPUT', that is not used actively in the program. OUTPUT is left in the program code for debugging purposes in future revisions. This subroutine prints whatever is in the accumulator to the screen. Thus, the programmer can compare what is there against what was expected. We used this subroutine heavily in writing the program code.

The parallel part of the program that coordinates with the MASTER section is SLAVE. In order for MASTER to operate correctly on the initiation end of data transfer, the receiving end must have a working copy of MICROLAN on his disk. The following documentation will be a description of how SLAVE works in conjunction with MASTER.

In order for the receive portion (SLAVE) of the program to be initiated, the receiving operator must initialize his copy of MICROLAN. As previously stated, the program is executed by typing the word "MICROLAN". The operator will then be prompted to identify which disk drive he desires to work from, (A,B,C, or default), and then be prompted for an "R" to initiate the execution.

The program begins by listening for an attention signal, which is an 'R' (ATTN) from the transmitting micro. This is used by the MASTER to see if someone is out there ready to accept data transfer. SLAVE continues to listen until an ATTN is received. Once it is received, a message string is printed to the screen to let the operator know that a connection was made. SLAVE then sends an 'r' (RXACK) to the MASTER to acknowledge receipt of the ATTN.

The same procedure is essentially repeated, only with a few changes. SLAVE now listens for a 'DC2' (TXSYM) and continues to listen until one is received. This is done for synchronization and acknowledgement that SLAVE is aware that data transfer is about to take place.

Before an acknowledgement signal is sent back to MASTER, a few operations will take place. This is done for synchronization.

The filename of the file that is being transferred is stored in a memory location known as the File Control Block, of FCB. The size is 32 spaces. The FCB is reset with zeroes to ensure that any previous data will not interfere. Once the FCB is reset, a 't' (TXACK) is sent to MASTER for acknowledgement that synchronization is set and SLAVE is ready for data reception.

The filename will be the first bit of information sent. Once SLAVE receives that first byte, it does a few comparisons before it writes it to memory.

First it checks for a 'DC4' (QUIT). If it receives one of these, it prints a message to the screen stating that no file transfer has taken place and then jumps out of the program (back to CPM). If the data was not a QUIT, then it is compared to a zero. A zero means that the filename has been completely sent and the program continues If it was not a zero, then the comparison is against a TXSYM. This is done to ensure that the data was valid. A few TXSYM's may have been sent over from MASTER after synchronization was established on the SLAVE end. This procedure is a safeguard against reading those extra TXSYM's as data. If one does get through, the program loops itself until valid data is received.

Once the filename data is received, it is put into the FCB memory location and then printed to the screen. This allows the operator to see which file is being sent. A checksum is calculated (see MASTER for explanation of method) throughout the reception of the filename for use later as a verification that the correct filename was received.

After the filename is received, an RXACK is transmitted to the MASTER to acknowledge that the filename has been received and SLAVE is awaiting the checksum calculated by MASTER. When this data is received, it is compared to the checksum calculated by SLAVE. If they are not the same, three (3) is added to the value sent by MASTER to announce that the checksums did not match. This means that the filename sent was not the same as the filename received. SLAVE then awaits for a re-transmission of the checksum + 3 sent previously. Once this is received, SLAVE acknowledges with a RXACK (which ensures synchronization), returns to reset the FCB and starts all over again, listening for a re-transmission of the filename.

If the checksums do match, then the program continues by sending a 'g' (GOOD). This verifies receipt of a good checksum. The subroutine OPNFILE then checks the directory to see if a file already exists by that filename which was previously transmitted. If one does exist, a QUIT is sent to MASTER advising that micro that a file already exists by that filename. A string is then printed to the screen telling the operator of the duplication of filenames, followed by the program jumping to CPM, terminating this session of SLAVE. If the filename did not previously exist in the directory, then a new file is created.

We are now ready to receive the data in 128-byte blocks. The Direct Memory Address (DMA) is a dedicated block of memory, 128 bytes long, used for this purpose. A synchronization check is done first and then a TXACK is sent to MASTER when SLAVE is ready to receive data. To separate the received data from MICROLAN's command and synchronization bytes, SLAVE now looks for a RLDTA symbol from MASTER. While SLAVE is looking for RLDTA, it also performs one other check. If a QUIT was sent, that signals the end of transmission and the file is closed. Once SLAVE sees a RLDTA symbol, it echoes back to MASTER, then proceeds to look for data. First, SLAVE checks to see that RLDTA is not still being sent by MASTER (only on the first byte of the 128-byte block). Once SLAVE is sure that the data block is being sent, it enters the data receive loop. The data byte is moved into memory and a checksum calculated for each run through this loop. This procedure continues until the counter, intialized with the size of the DMA, has reached zero. This indicates that 128 bytes of data has been sent. MASTER sends its checksum and SLAVE compares it with its own. If it does not match, SLAVE sends a 'b' (BAD) to MASTER indicating that it must re-transmit the same 128 bytes. If the checksums do agree, then the 128 bytes are written to the disk and an asterisk is printed to the screen telling the operator that 128 bytes of data have been successfully transferred. The program then returns to repeat this process until a QUIT is received.

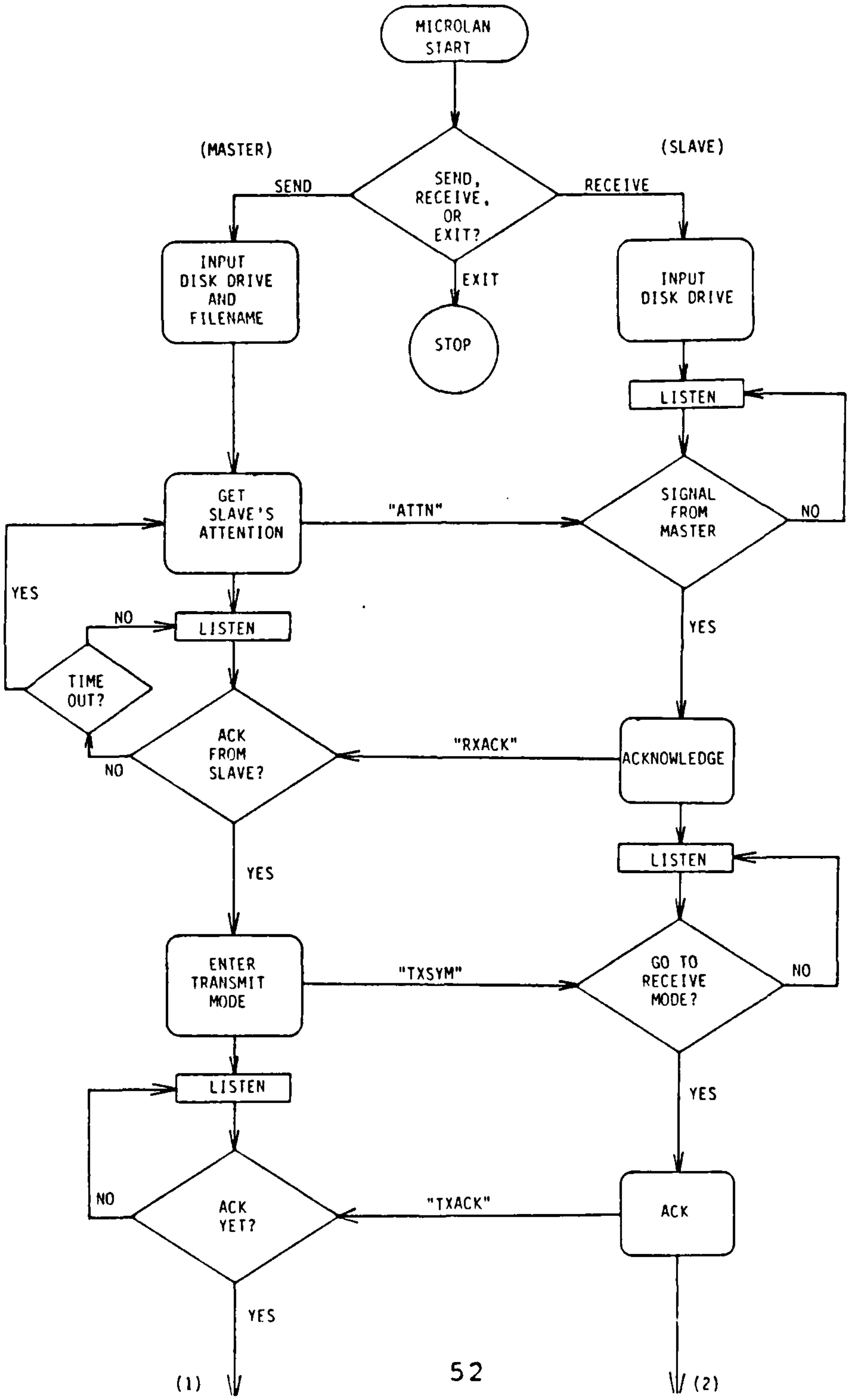
When a QUIT is received, SLAVE acknowledges by sending a QUIT back and then closes the file. A string is printed on the screen indicating to the operator that file transmission is complete. SLAVE then waits for a 'Z' (DONE) from MASTER which ensures that the session is complete. A DONE is transmitted back which completes the hand-shaking process and then SLAVE jumps to CPM. The SLAVE program has been terminated and the micro is ready for any command. If the operator wishes to receive another file, he must reinitiate the MICROLAN program.

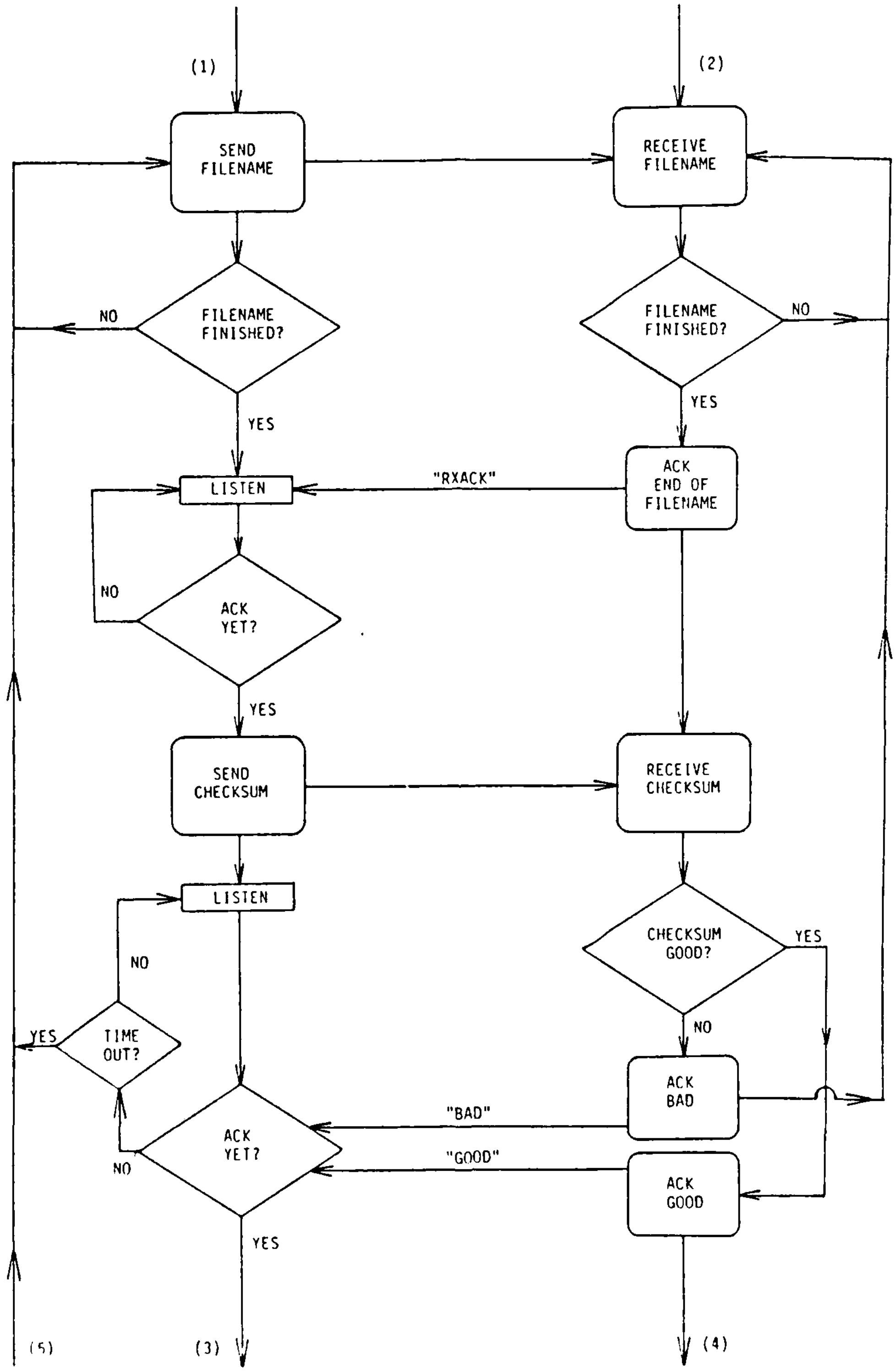
There are two safety factors that are included in the SLAVE program that were not previously mentioned. The first one concerns the occurrence of a full disk on the part of the receiving micro Each time the program writes a 128-byte block of data to the disk, it checks to see if the disk is full. In the event of a full disk, SLAVE sends a 'd' (DSKFUL) to MASTER expressing that there is no more room on the disk and cannot receive any more data. SLAVE then awaits confirmation from MASTER that it has received the DSKFUL. Confirmation is acknowledged by the receipt of a DONE, which completes the "handshaking". A string is printed to the screen letting the operator know that he received an incomplete file due to a full disk. SLAVE then goes to CPM.

The other safety factor handles the occurrence of the transmitting micro aborting file transfer. To abort file transfer, the operator of the transmitting file uses a "control C". SLAVE listens for this "control C" throughout the entire program. Every time data is received from MASTER, it is checked for the abort signal. This allows for the option of the operator at the transmitting micro to stop data transfer at any time. If this happens, the program goes to a subroutine which sends a "control C" back to MASTER in acknowledgement and then prints a string to the screen. This tells the operator that file transfer has been aborted and that no file will exist under the filename that was passed. The program then jumps to CPM. Our logic was based on "whole file or no file". We felt that having an empty file would be an unmistakable indicator that the file transfer was incomplete and that retransmission was necessary. If the operator wishes to retain a partial file, a minor change to the program would be needed. The file would have to be closed before the program jumped to CPM by invoking the subroutine "CLSFILE" first.

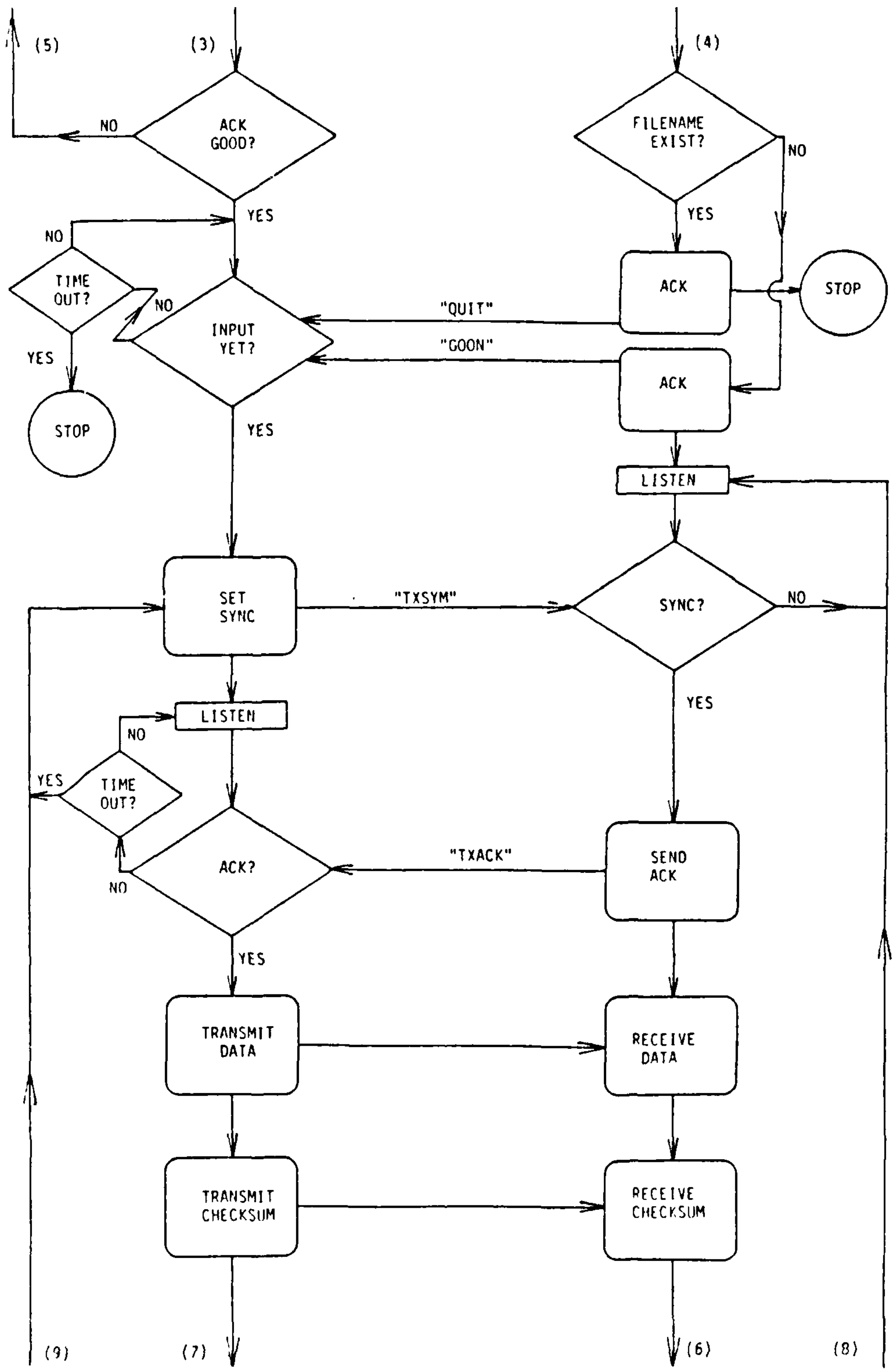
## APPENDIX B

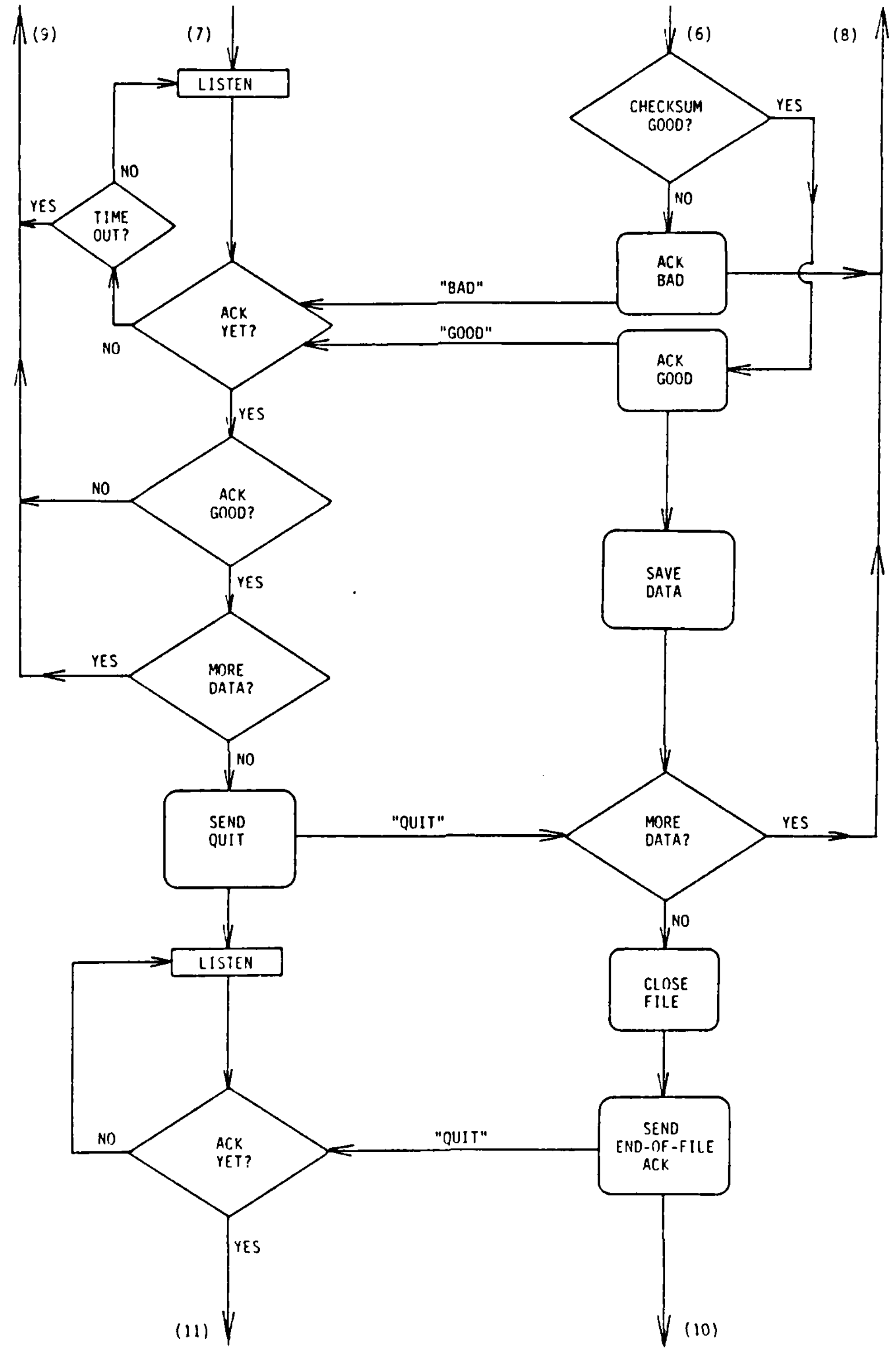
## FLOW DIAGRAM



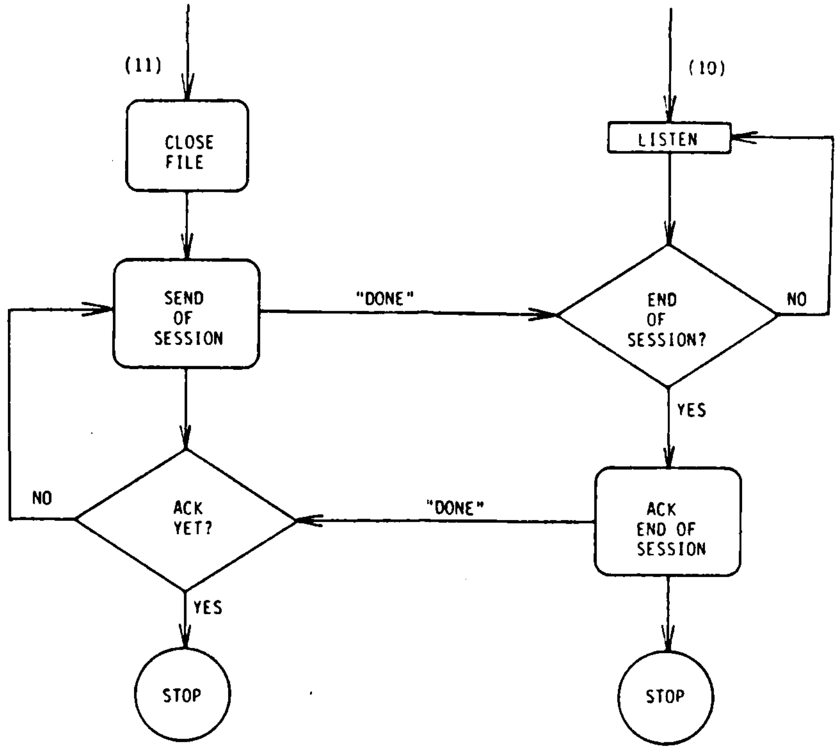


(1)
(2)
SEND FILENAME
RECEIVE FILENAME
FILENAME FINISHED?
NO
YES
FILENAME FINISHED?
NO
YES
LISTEN
"RXACK"
ACK END OF FILENAME
NO
ACK YET?
YES
SEND CHECKSUM
RECEIVE CHECKSUM
LISTEN
NO
CHECKSUM GOOD?
YES
NO
YES
TIME OUT?
ACK BAD
"BAD"
ACK YET?
NO
"GOOD"
ACK GOOD
YES
(5)
(3)
(4)


(5)
(3)
(4)
ACK GOOD?
NO
YES
FILENAME EXIST?
NO
YES
NO
TIME OUT?
YES
STOP
NO
INPUT YET?
YES
ACK
"QUIT"
STOP
"GOON"
ACK
LISTEN
SET SYNC
"TXSYM"
SYNC?
NO
YES
LISTEN
NO
YES
TIME OUT?
NO
ACK?
"TXACK"
SEND ACK
YES
TRANSMIT DATA
RECEIVE DATA
TRANSMIT CHECKSUM
RECEIVE CHECKSUM
(9)
(7)
(6)
(8)



(9)
(7)
(6)
(8)
LISTEN
CHECKSUM GOOD?
YES
NO
YES
TIME OUT?
NO
ACK BAD
"BAD"
ACK YET?
"GOOD"
ACK GOOD
NO
YES
ACK GOOD?
NO
YES
SAVE DATA
MORE DATA?
YES
NO
SEND QUIT
"QUIT"
MORE DATA?
YES
NO
LISTEN
CLOSE FILE
ACK YET?
NO
"QUIT"
SEND END-OF-FILE ACK
YES
(11)
(10)


(11)
CLOSE FILE
SEND OF SESSION
"DONE"
NO
ACK YET?
YES
STOP
(10)
LISTEN
END OF SESSION?
NO
YES
ACK END OF SESSION
"DONE"
STOP

# APPENDIX C

## CP/M - 80

```
TRUE      EQU 0FFFFH
NORTHSTAR EQU TRUE
APPLE     EQU NOT TRUE
          ORG 0100H
INIT      LXI SP,0A000H        ;MOVE STACK PTR TO SAFE LOC
          CALL CRLF
          MVI C,09H            ;PRINT STRING TO SCREEN
          LXI D,RIGHTS         ;COPYRIGHTS AND NAMES OF AUTHORS
          CALL BDOS
          CALL CRLF
          CALL CRLF
          MVI C,09H            ;PRINT STRING TO SCREEN
          LXI D,WELCUM         ;WELCOME MSG
          CALL BDOS
          CALL CRLF
          CALL CRLF
          MVI C,09H            ;PRINT STRING TO SCREEN
          LXI D,INSTRC         ;SEND, RECEIVE, OR QUIT?
          CALL BDOS
          CALL CRLF
          CALL CRLF
```

```
HOLDING MVI C,06H           ;CHECK FOR CONSOLE INPUT
        MVI E,0FFH          ;LOOKING FOR INPUT
        CALL BDOS
        ANI 0DFH            ;ENSURE LETTER IS A CAPITAL
        CPI 53H             ;IS IT AN 'S'?
        JZ MASTER           ;IF SO, START FILE TRANSFER
        CPI 52H             ;IS IT AN 'R'?
        JZ SLAVE            ;IF SO, PREPARE TO RECEIVE FILE
        CPI 58H             ;IS IT AN 'X'?
        JZ CPM              ;IF SO, GO TO CPM
        JMP HOLDING         ;REPEAT UNTIL INPUT FOUND
MASTER  MVI C,09H           ;PRINT STRING TO SCREEN
        LXI D,ENTER         ;ENTER FILENAME
        CALL BDOS
        CALL CRLF
        CALL CRLF
FILLUP  LXI H,FCB           ;ADDRESS OF FCB
        MVI M,00H
        INX H
        MVI C,0BH           ;11 SPACES
FILLUP1 MVI M,20H           ;FILL MEMORY ADDRESS WITH SPACES
        INX H               ;MOVE PTR TO NEXT ADDRESS
        DCR C               ;DECREMENT COUNTER
        JNZ FILLUP1         ;REPEAT UNTIL DONE
```

```
        MVI C,13H         ;TOTAL OF 20 SPACES
FILLUP2 MVI M,00H         ;FILL REST OF ADDRESS WITH 0'S
        INX H             ;MOVE PTR TO NEXT ADDRESS
        DCR C             ;DECREMENT COUNTER
        JNZ FILLUP2       ;REPEAT UNTIL DONE
HOLD1   MVI C,0AH         ;READ CONSOLE BUFFER
        LXI D,CONBUF      ;ADDRESS OF FIRST LETTER OF FILENAME
        CALL BDOS
        LXI H,CONBUF      ;ADDRESS OF CONSOLE BUFFER
        LXI D,FCB+1       ;FCB ADDRESS
        INX H
        MOV B,M           ;STORE COUNT IN B REGISTER
        MOV A,M           ;MOVE COUNT TO ACCUMULATOR
        ORA A             ;IS THERE AN INPUT?
        JZ ERROR          ;TRY AGAIN
        INX H
FLUP    MOV A,M
        CPI 3AH           ;IS CHARACTER A ':'?
        JZ DSKSEL         ;IF SO, GO TO DISK SELECT
        CPI 2EH           ;IS IT A '.'?
        JZ FIXIT          ;IF SO, SKIP TO FILETYPE
        CPI 40H           ;CHECK FOR LETTER
        JC DONTFIX        ;SKIP NEXT STEP IF NOT LETTER
        ANI 0DFH          ;ENSURE LETTER IS A CAPITAL
```

```
DONTFIX STAX D                  ;STORE CHARACTER IN FCB
        INX D
        INX H
        DCR B
        JNZ FLUP                ;REPEAT UNTIL END
LISN1   MVI A,ATTN              ;LETTER 'R'
        CALL POUT1
        MVI C,03H               ;LISTEN 3 TIMES
LISN    CALL PIN
        CPI RXACK               ;IS IT AN 'r'
        JZ XMIT                 ;IF SO, THEN XMIT
        DCR C                   ;OTHERWISE DECREMENT CTR
        JNZ LISN                ;LISTEN UNTIL CTR IS ZERO
        JMP LISN1               ;THEN TRY AGAIN
XMIT    CALL SWAP
        CALL CRLF
        CALL CRLF
        MVI C,09H               ;PRINT STRING TO SCREEN
        LXI D,RXING1            ;AN 'r' WAS RECEIVED
        CALL BDOS
        CALL CRLF
        CALL SWAP
XMIT1   MVI A,TXSYM             ;DC2 SYMBOL FOR SYNC AT START
        CALL POUT1              ;OF 128 BYTE BLOCK
```

```
        MVI C,08AH          ;LISTEN 138 TIMES
LITTLET CALL PIN
        CPI TXACK           ;WAS 't' RECEIVED?
        JZ TXFCB            ;IF SO, XMIT FILE CTRL BLK
        DCR C               ;OTHERWISE KEEP LISTENING
        JNZ LITTLET         ;UNTIL CTR IS ZERO,
        JMP XMIT1           ;THEN SEND DC2 SYNC AGAIN
TXFCB   CALL CRLF
        CALL OPENIT         ;SEE IF FILE EXISTS.  IF SO, OPEN IT
TXFCB1  MVI B,00H           ;INITIALIZE CHECKSUM REGISTER
        LXI H,FCB           ;SET PTR TO 1ST LETTER IN FILENAME
FCBLUP  MOV A,B             ;PERFORM CHECKSUM OPERATION
        INX H               ;(MOVE PTR TO NEXT BYTE)
        XRA M               ;BY XORING CURRENT BYTE
        MOV B,A             ;WITH B REGISTER
        MOV A,M             ;PUT CURRENT BYTE IN ACCUM
        CALL POUT1          ;SEND CURRENT BYTE
        CPI 0H              ;CHECK FOR END OF FILENAME
        JZ FCBCK            ;IF END, GO TO CHECKSUM LOOP
        JMP FCBLUP          ;IF NOT, REPEAT FCB LOOP
FCBCK   MVI C,20H           ;LOOP 32 TIMES
FCBCK1  CALL PIN            ;FOR SYNC WITH SLAVE
        CPI RXACK           ;IS IT AN 'r'?
        JNZ FCBCK1          ;IF NOT, LISTEN AGAIN.  IF SO,
```

```
        MOV A,B             ;PUT CHECKSUM IN ACCUM
        CALL POUT           ;SEND CHECKSUM
        PUSH B              ;SAVE CHECKSUM
        MVI A,0H            ;CLEAR ACCUM
        MVI B,80H           ;LISTEN 100 TIMES
FCBTMCT CALL PIN            ;READ MAIL
        CPI BAD             ;DID IT CHECK BAD?
        JZ RSNDFCB          ;IF SO, SEND FCB AGAIN
        CPI GOOD            ;DID IT CHECK GOOD?
        JZ WAITFIL          ;IF SO, GO TO NEXT ROUTINE
        DCR B               ;IF NOT, DECREMENT CTR, AND
        JNZ FCBTMOT         ;IF NOT 0, LISTEN AGAIN
        POP B               ;CLEAR STACK
        DCR C               ;IF SO, DECREMENT C
        JNZ FCBCK1          ;AND REPEAT UNTIL C=0
        JMP TXFCB           ;IF 0, ASSUME PROBLEM AND SEND AGAIN
WAITFIL POP B               ;CLEAR STACK
WAIT2   LXI B,07FFH         ;COUNT LOOP APPX 2K
WAIT1   CALL STATIN1        ;ANY 'MAIL'?
        JZ WAIT1            ;IF NOT, CHECK AGAIN
        DCX B               ;IF SO, DECREMENT CTR
        MOV A,B
        CRA C
        JZ GOCPM            ;AND, IF 0, QUIT
```

```
        CALL PIN            ;OTHERWISE READ 'MAIL'
        CPI QUIT            ;DOES RXING MICRO ALREADY HAVE FILE?
        JZ GOCPM1           ;IF SO, GO TO CPM
        CPI GOON            ;IS IT THE GO ON SIGNAL 'G'
        JNZ WAIT2           ;IF NOT, LISTEN AGAIN.  ALLOW RXING
                            ;MICRO TO CATCH UP
        CALL SWAP
        CALL CRLF
        CALL SWAP
TXDATA  CALL SWAP           ;SEND THE FILE
        MVI C,09H           ;PRINT STRING TO SCREEN
        LXI D,TXING1        ;SAYS FILE BEING SENT
        CALL BDOS
        CALL CRLF
        CALL SWAP
RDSEQ   CALL READSEQ        ;READ FIRST 128 BYTE BLOCK
SEND    CALL CHECK          ;AND SEND TO RXING MICRO
        MVI A,TXSYM         ;DC2 SYMBOL FOR SYNC AT START OF DATA
        CALL POUT1
        MVI C,0FH           ;LISTEN 15 TIMES
LITLET2 CALL PIN
        CPI TXACK           ;IS IT A 't'
        JZ SLUP2            ;IF SO, READY TO SEND DATA
        CPI DSKFUL          ;IS RXING MICRO'S DISK FULL?
```

```
        JZ FULDISK          ;IF SO, QUIT
        DCR C               ;IF NOT, DECREMENT CTR
        JNZ LITLET2         ;LISTEN AGAIN, UNLESS CTR IS 0,
        JMP SEND            ;THEN TRY TO SYNC AGAIN
SLUP2   MVI A,RLDTA         ;0CBH MEANS TIME FOR DATA
        CALL POUT1
        LXI B,07FFH         ;WAIT LOOP APPX 2K
SLUP3   CALL PIN
        CPI RLDTA           ;IS IT ECHO?
        JZ SLUP1            ;IF SO, SEND DATA
        DCX B               ;DECREMENT COUNTER
        MOV A,B
        ORA C
        JNZ SLUP3           ;REPEAT UNTIL ZERO
        JMP SLUP2           ;THEN SEND AGAIN
SLUP1   LXI H,DMA           ;POINTER TO 1ST INFO BYTE
        MVI B,00H           ;INITIALIZE CHECKSUM
SLOOP   MOV A,B             ;PERFORM CHECKSUM
        XRA M               ;XOR DATA WITH B REGISTER
        MOV B,A
        MOV A,M             ;PUT BYTE IN ACCUMULATOR
        CALL POUT           ;DATA IS TRANSFERRED
        INX H               ;MOVE PTR TO NEXT BYTE
        MOV A,H             ;MOVE H REG TO ACCUM
```

```
         CPI ENDMA              ;END DMA, CHECK FOR LAST BYTE
         JNZ SLOOP              ;IF NOT, SEND NEXT BYTE.  OTHERWISE
CRC      MOV A,B                ;PUT CHECKSUM IN ACCUMULATOR
         CALL POUT              ;AND SEND TO RXING MICRO
CRCTMOT  MVI B,01AH             ;LISTEN 26 TIMES
CRCT1    CALL STATIN1           ;CHECK INPUT BUFFER
         JZ CRCT1               ;IF NOTHING, TRY AGAIN
         CALL PIN               ;READ MAIL
         CPI BAD                ;IS CHECK BAD?
         JZ RESEND              ;IF SO, SEND BLOCK AGAIN
         CPI GOOD               ;IS CHECK GOOD?
         JZ RDSQRPT             ;IF SO, READ NEXT BLOCK
         DCR B                  ;DECREMENT COUNTER
         JNZ CRCT1              ;IF NOT TIMED OUT,LISTEN AGAIN
         JMP SEND               ;IF TIMED OUT, ASSUME PROBLEM, AND
                                ;SEND BLOCK AGAIN
DSKSEL   CALL SWAP
         LXI H,CONBUF+2         ;ADDRESS OF DISK SEL ENTRY
         MOV A,M                ;PUT DISK SEL IN ACCUM
         ANI ODFH               ;ENSURE LETTER IS CAPITAL
         CPI 'A'                ;IS LETTER AN 'A'?
         JZ ADISK               ;IF SO, SET FOR A DRIVE.
         CPI 'B'                ;IS LETTER A 'B'?
         JZ BDISK               ;IF SO, SET FOR B DRIVE.
```

```
        JNZ DSKSEL1            ;IF NEITHER, RETURN TO FILENAME LOOP.
ADISK   LXI H,FCB              ;SET PTR TO DRIVE BYTE.
        MVI M,01H              ;SET FCB FOR A DRIVE.
        JMP DSKSEL1            ;RETURN TO FILENAME LOOP.
BDISK   LXI H,FCB              ;SET PTR TO DRIVE BYTE.
        MVI M,02H              ;SET FCB FOR B DRIVE.
DSKSEL1 CALL SWAP
        LXI H,CONBUF+4         ;MOVE BUFFER POINTER TO FILENAME.
        LXI D,FCB+1            ;FCB FILENAME ADDRESS.
        JMP FLUP               ;RETURN TO FILENAME LOOP.
RESEND  CALL SWAP
        MVI C,CONOUT           ;PRINT TO SCREEN
        MVI E,BAD              ;A 'b' IF CHECKSUM WAS BAD
        CALL BDOS
        CALL SWAP
        JMP SEND               ;AND SEND BLOCK AGAIN
RSNDFCB POP B                  ;RECALL CHECKSUM
        MOV A,B                ;PUT CHECKSUM IN ACCUM
        ADI 3                  ;ADD 3 TO OFFSET
        CALL POUT1             ;SEND BYTE
RSNDF   CALL PIN
        CPI FYACK              ;IS IT AN 'r'?
        JNZ RSNDF              ;IF NOT, LISTEN AGAIN
RSNDFC1 CALL PIN               ;READ MAILBOX
```

```
        CPI TXACK               ;SYNC WITH RXING MICRO
        JNZ RSNDFC1             ;REPEAT UNTIL TXACK RECEIVED
        JMP TXFCB1              ;IF SO, RESEND FCB
FIXIT   LXI D,FCB+9             ;MOVE POINTER TO FILETYPE AREA
        INX H                   ;MOVE PTR TO FIRST LETTER OF FILTYP
        JMP FLUP
ERROR   MVI C,09H               ;PRINT STRING TO SCREEN
        LXI D,ERMSG             ;ERROR MESSAGE
        CALL BDOS
        CALL CRLF
        JMP HOLD1               ;LOOK FOR INPUT AGAIN
POUT1   PUSH PSW                ;SEND THE DATA
                                ;FIRST, SAVE THE CURRENT BYTE
        MVI C,06H               ;CHECK FOR CONSOLE INPUT
        MVI E,0FFH              ;LOOKING FOR INPUT
        CALL BDOS
        CPI CTRLC               ;IS THERE A CONTROL C?
        JZ STOPS                ;IF SO, ABORT; OTHERWISE,
        CALL CHECK              ;PERFORM CHECK
        POP PSW                 ;AND RECALL BYTE
        IF NORTHSTAR            ;IF MICRO IS NORTHSTAR
         OUT DATA               ;THEN XMIT BYTE
        ENDIF
        IF APPLE                ;IF MICRO IS APPLE
```

```
         STA DATA1            ;XMIT BYTE
         ENDIF
         RET
OPENIT   MVI C,0FH            ;OPEN FILE CODE
         LXI D,FCB            ;FILE CTRL BLOCK ADDRESS IN DE REG PR
         CALL BDOS
         CPI 0FFH             ;FF = FILE NOT FOUND
         JZ FNFOUND           ;IF FILE NOT FOUND
         RET                  ;OTHERWISE, RET TO TX DATA
CLOSIT   MVI C,10H            ;CLOSE FILE CODE
         LXI D,FCB            ;FILE CTRL BLOCK ADDRESS IN DE REG PR
         CALL BDOS
CLOSIT1  MVI A,DONE           ;END OF SESSION MSG 'Z'
         CALL POUT1           ;SEND TO RXING MICRO
         MVI A,0H             ;CLEAR ACCUM
         CALL PIN             ;CHECK REPLY
         CPI DONE             ;DOES RXING MICRO AGREE?
         JNZ CLOSIT1          ;IF NOT, REPEAT
         JMP GOCPM            ;IF SO, GO TO CPM
READSEQ  MVI C,14H            ;READ SEQUENTIAL CODE
         LXI D,FCB            ;FILE CTRL BLOCK ADDRESS IN DE REG PR
         CALL BDOS
         CPI 0                ;0 MEANS SUCCESSFUL READ
         JNZ EOFILE           ;IF NOT 0, ASSUME FINISHED WITH FILE
```

```
        RET
RDSQRPT CALL SWAP
        MVI C,CONOUT          ;PRINT TO SCREEN
        MVI E,02AH            ;'*' SO USER KNOWS BLK WAS SENT
        CALL BDOS
        CALL SWAP
        JMP RDSEQ             ;TO READ NEXT 128 BYTE BLK
FNFOUND MVI A,QUIT            ;TELL RXING MICRO NO FILE FOUND
        CALL POUT1
        MVI C,09H             ;PRINT STRING TO SCREEN
        LXI D,FNFDMSG         ;FILE NOT FOUND MSG
        CALL BDOS
        CALL CRLF
        JMP GOCPM             ;AND GO TO CPM
EOFILE  POP PSW               ;CORRECT STACK POINTER
EOFILE2 MVI A,TXSYM           ;DC2 SYMBOL FOR SYNC WITH RXING MICRO
        CALL POUT1
        MVI C,0FH             ;LISTEN 15 TIMES
LITLET3 CALL STATIN1          ;CHECK FOR MAIL
        JZ LITLET3            ;IF NONE, CHECK AGAIN
        CALL PIN              ;READ MAIL
        CPI TXACK             ;IS IT A 't'?
        JZ EOFIL1             ;IF SO, CONTINUE
        DCR C                 ;IF NOT, DECREMENT COUNTER
```

```
        JNZ LITLET3         ;AND LISTEN AGAIN, UNLESS COUNTER IS
        JMP EOFILE2         ;0.  THEN TRY AGAIN
EOFIL1  MVI A,QUIT          ;DC4 SYMBOL.  TELLS RXING MICRO THAT
        CALL POUT1          ;THE FILE IS DONE
        CALL PIN            ;LISTEN FOR REPLY
        CPI QUIT            ;DOES RXING MICRO ACKNOWLEDGE?
        JNZ EOFIL1          ;IF NOT, TRY AGAIN
        CALL CRLF
        MVI C,09H           ;PRINT STRING TO SCREEN
        LXI D,EOFMSG        ;IF SO, TELL USER FILE IS DONE
        CALL BDOS
        CALL CRLF
        JMP CLOSIT          ;AND CLOSE THE FILE
STOPS   MVI A,CTRLC         ;SEND CTRLC TO RXING MICRO
        IF NORTHSTAR        ;IF NORTHSTAR MICRO
         OUT DATA           ;SEND CONTROL C
        ENDIF
        IF APPLE            ;IF APPLE MICRO
         STA DATA1          ;SEND CONTROL C
        ENDIF
        MVI A,0H            ;CLEAR ACCUM
        CALL PIN            ;FROM RXING MICRO
        CPI CTRLC           ;ACK FROM RXING MICRO
        JNZ STOPS           ;REPEAT UNTIL ACK
```

```
         POP PSW                   ;CORRECT STACK POINTER
         JMP GOCPM
FULDISK  MVI A,DONE                ;LETTER 'Z' TO ACKNOWLEDGE
         CALL POUT1                ;SEND BYTE
         MVI C,09H                 ;PRINT STRING TO SCREEN
         LXI D,FULMSG              ;SAYS RXER'S DISK FULL
         CALL BDOS
         CALL CRLF
         JMP GOCPM
GOCPM    MVI A,0H                  ;RESET THE ACCUMULATOR AND
         IF NORTHSTAR              ;NORTHSTAR MICRO
          OUT DATA                 ;SEND BYTE
         ENDIF
         IF APPLE                  ;APPLE MICRO
          STA DATA1                ;SEND BYTE
         ENDIF
         CALL CRLF
         CALL PIN
         JMP CPM                   ;AND GO TO CPM
GOCPM1   MVI C,09H                 ;PRINT STRING TO SCREEN
         LXI D,HASFILE             ;RXING MICRO HAS FILE ALREADY
         CALL BDOS
         JMP GOCPM
SLAVE    MVI C,09H                 ;PRINT STRING TO SCREEN
```

```
        LXI D,WCHDSK        ;SELECT DISK DRIVE
        CALL BDOS
        CALL CRLF
        CALL CRLF
DRVSEL  MVI C,06H           ;CHECK FOR CONSOLE INPUT
        MVI E,0FFH          ;LOOKING FOR INPUT
        CALL BDOS
        CPI 0DH             ;IS IT A <CR>?
        JZ CONT             ;IF SO, ENTER RECEIVE MODE
        ANI 0DFH            ;ENSURE LETTER IS A CAPITAL
        CPI 'A'             ;IS IT AN 'A'?
        JNZ DISKB           ;SKIP TO B IF NOT 'A'
        LXI H,FCB           ;ADDRESS OF DISK DRIVE BYTE
        MVI M,01H           ;SET BYTE TO A DISK DRIVE
        JMP CONT            ;THEN CONTINUE
DISKB   CPI 'B'             ;IS IT A 'B'?
        JNZ DRVSEL          ;IF NOT, CONTINUE IN RECEIVE MODE
        LXI H,FCB           ;ADDRESS OF DISK DRIVE BYTE
        MVI M,02H           ;SET BYTE TO B DISK DRIVE
CONT    MVI C,09H           ;PRINT STRING TO SCREEN
        LXI D,RXMODE        ;IN RECEIVE MODE
        CALL BDOS
        CALL CRLF
SLAVE1  MVI A,00H           ;RESET ACCUMULATOR
```

```
        CALL PIN            ;LISTENING FOR AN 'R'
        CPI ATTH            ;'R'
        JNZ SLAVE1          ;IF 'R' RX'D, CONTINUE. IF NOT
                            ;LISTEN AGAIN
        CALL SWAP
        CALL CRLF
        MVI C,09H           ;PRINT STRING TO SCREEN
        LXI D,RXING1        ;CONNECTION MADE
        CALL BDOS
        CALL SWAP
        CALL CRLF
        MVI A,RXACK         ;'r'
        CALL POUT           ;SEND A 'r' TO XMITING MICRO
LISTEN  CALL PIN1           ;LISTENING FOR A 'DC2'
        CPI TXSYM           ;'DC2'
        JNZ LISTEN          ;IF 'DC2' RX'D, CONTINUE. IF NOT,
                            ;LISTEN AGAIN
        CALL CRLF
RXFCB   CALL SWAP
        LXI H,FCB+1         ;ADDRESS OF FCB MEM LOC INTO
                            ;H,L REGISTER PAIR
        MVI C,1EH           ;COUNTER FOR FCB'S 31 SPACES
RSTFCB  MVI M,00H           ;FILL FCB WITH 0'S
        INX H               ;MOVE PTR TO NEXT MEMORY ADDRESS IN FCB
```

```
        DCR C                   ;DECREMENT COUNTER
        MOV A,C
        CPI 0
        JNZ RSTFCB              ;IF COUNTER = 0, CONT. IF NOT,
                                ;PUT ANOTHER 0 IN FCB
        CALL SWAP
        MVI B,00H               ;INITIALIZE CHECKSUM
        LXI H,FCB+1             ;LOAD 2ND ADDRESS OF FCB IN
                                ;H,L REGISTER PAIR
        MVI A,TXACK             ;'t'
        CALL POUT               ;SEND 't' TO XMITING MICRO FOR SYNC
        CALL PIN1               ;CLEAR THE ACCUMULATOR
RST1    CALL STATIN1            ;CHECKING FOR INPUT
        JZ RST1
RST2    CALL PIN1               ;FILE NAME DATA
        CPI QUIT                ;IS DATA A 'QUIT'?
        JZ NOFILE               ;FILE DID NOT EXIST
        CPI 0H                  ;CHECK IF FILENAME COMPLETELY SENT
        JZ FCBCRC               ;IF FILENAME RX'D, GO TO CHECKSUM
        CPI TXSYM               ;CHECK IF DATA IS VALID
        JZ RST2                 ;IF DATA IS NOT FILENAME,
                                ;CHECK NEXT BYTE
        MOV M,A                 ;PUT FILENAME IN FCB
        CALL OUTPUT             ;PRINT FILENAME TO SCREEN
```

```
        MOV A,B
        XRA M               ;CALCULATE CHECKSUM
        MOV B,A
        INX H               ;MOVE PTR TO NEXT FCB ADDRESS
        JMP RST1
FCBCRC  MVI A,RXACK         ;'r'
        CALL POUT           ;SYNC DATA WITH XMITING MICRO
FCBCRC1 CALL STATIN1        ;CHECKING FOR INPUT
        JZ FCBCRC1
        CALL CRLF
        CALL PIN            ;CHECKSUM DATA
        CMP B               ;COMPARE CHECKSUM
        JZ STRTFIL          ;CHECKSUM MATCHED
        ADI 3               ;ADD 3 TO THE CHECKSUM
        MOV C,A             ;STORE IN REGISTER
        MVI A,BAD           ;CHECKSUM DID NOT MATCH
        CALL POUT           ;TELL XMITING MICRO
CLEAR   CALL PIN
        CMP C               ;XMITING MICRO STOPPED SENDING CHKSUM?
        JNZ CLEAR           ;IF NOT,LISTEN AGAIN
        MVI A,RXACK         ;SYNC WITH XMITING MICRO
        CALL POUT
        JMP EXFCB           ;TRY AGAIN
STRTFIL MVI A,GOOD          ;READY TO CHECK IF FILENAME ALREADY USED
```

```
        CALL POUT
        CALL OPNFILE        ;CHECK IF FILENAME EXISTS
        CALL MAKEFIL        ;CREATE NEW FILE
RXD1    MVI B,00H           ;INITIALIZE CHECKSUM
        LXI H,DMA           ;LOAD ADDRESS OF DMA MEM LOC TO
                            ;H,L REGISTER PAIR
        MVI C,81H           ;INITIALIZE COUNTER WITH SIZE OF DMA
RXDS    CALL STATIN1        ;CHECKING FOR INPUT
        JZ RXDS
RXD2    CALL PIN1           ;SYNC WITH XMITING MICRO
        CPI TXSYM           ;COMPARE WITH 'DC2'
        JNZ RXD2
        MVI A,TXACK         ;'t'
        CALL POUT           ;IN SYNC WITH XMITING MICRO
RXDS1   CALL STATIN1        ;CHECKING FOR INPUT
        JZ RXDS1
RXYET   CALL PIN1
        CPI RLDTA           ;IS IT 0CBH?
        JZ RXYET1           ;IF SO, GO TO RECEIVE DATA
        CPI QUIT            ;IS IT 'DC4' FOR QUIT?
        JZ CLSFILE          ;IF SO, CLOSE FILE; OTHERWISE,
        JMP RXDS1           ;LISTEN AGAIN
RXYET1  MVI A,RLDTA         ;ACK REAL DATA COMING
        CALL POUT
```

```
        MVI A,00H           ;CLEAR ACCUM
RXYET2  CALL STATIN1        ;CHECK FOR INPUT
        JZ RXYET2
RXYET3  CALL PIN            ;READ DATA
        CPI RLDTA           ;IS IT STILL RLDTA?
        JZ RXYET3
RXD3    DCR C               ;DECREMENT COUNTER
        JZ RXCRC            ;CHECKSUM RX'D
        MOV M,A             ;PUT THE DATA IN MEMORY
        MOV A,B
        XRA M               ;CALCULATE CHECKSUM
        MOV B,A
        INX H               ;MOVE PTR TO NEXT DMA ADDRESS
RXD4    CALL STATIN1        ;CHECK FOR INPUT
        JZ RXD4             ;LOOP UNTIL INPUT
        CALL PIN
        JMP RXD3
RXCRC   MOV A,A             ;ENSURE B IS COMPARED TO A
        CMP B               ;COMPARE WITH CHECKSUM
        JZ WRITFIL          ;128 BYTE BLOCK SENT
        MVI A,BAD           ;CHECKSUM DID NOT MATCH
        CALL POUT           ;NOTIFY XMITING MICRO
        JMP RXD2            ;SEND 128 BYTE BLOCK AGAIN
POUT    PUSH PSW            ;SAVE THE DATA
```

```
        CALL CHECK
        POP PSW              ;RETURN THE DATA
        IF NORTHSTAR         ;MICRO IS NORTHSTAR
         OUT DATA            ;SENDS DATA ACROSS THE LINE
        ENDIF
        IF APPLE             ;MICRO IS APPLE
         STA DATA1           ;SENDS DATA ACROSS THE LINE
        ENDIF
        RET
PIN1    IF NORTHSTAR         ;MICRO IS NORTHSTAR
         IN DATA
        ENDIF
        IF APPLE             ;MICRO IS APPLE
         LDA DATA1
        ENDIF
        CPI CTRLC            ;DID XMITING MICRO ABORT?
        JZ ABORT             ;IF SO, ABORT
        RET
WRITFIL MVI A,GOOD           ;XMIT THAT THE CHECKSUM IS CORRECT
        CALL POUT
        CALL WRITSEQ         ;START WRITING FILE TO DISK
        CALL SWAP            ;SAVE REGISTERS
        MVI C,CONOUT         ;PRINT TO SCREEN
        MVI E,02AH           ;'*' TO PRINT TO SCREEN
```

```
        CALL BDOS
        CALL SWAP               ;RETURN REGISTERS
        JMP RXD1
CPNFILE MVI C,0FH               ;OPEN FILE CODE
        LXI D,FCB               ;FCB ADDRESS IN D,E RGSTR PAIR
        CALL BDOS
        CPI 0FFH                ;FF = FILE NOT FOUND
        JNZ FILFND              ;FILE EXISTS
        RET
CLSFILE MVI A,QUIT              ;'DC4'
        CALL POUT               ;AGREE END OF FILE
        MVI C,10H               ;CLOSE FILE CODE
        LXI D,FCB               ;FCB ADDRESS IN D,E RGSTR PAIR
        CALL BDOS
        CALL CRLF
        LXI D,EOFMSG            ;FILE TRANSMISSION COMPLETED
        MVI C,09H               ;PRINT STRING TO SCREEN
        CALL BDOS
        CALL CRLF
        MVI A,0H                ;CLEAR THE ACCUMULATOR
CLSFIL1 CALL PIN1               ;LOOKING FOR END OF SESSION MSG
        CPI DONE                ;'Z' = END OF SESSION
        JNZ CLSFIL1
        MVI A,DONE              ;END OF SESSION MESSAGE
```

```
        CALL POUT          ;CCNFIRM RECEPTION OF E-O-SESSION MSG
        JMP CPM
MAKEFIL MVI C,16H          ;MAKE NEW FILE CODE
        LXI D,FCB          ;FCB ADDRESS IN D,E RGSTR PAIR
        CALL BDOS
        MVI A,GOON         ;CCNTINUE MESSAGE
        CALL POUT
        RET                ;RETURN TO RX FIRST 128 BYTE BLOCK
WRITSEQ MVI C,15H          ;WRITE THE FILE TO THE DISK
        LXI D,FCB          ;FCB IN D,E RGSTR PAIR
        CALL BDOS
        ORA A              ;CHECK IF DISK IS FULL
        JNZ FULLDSK        ;IF SO, JUMP TO FULLDSK
        RET
FILFND  MVI A,QUIT         ;TELL XMITING MICRO, FILE FOUND
        CALL POUT
        MVI C,09H          ;PRINT STRING TO SCREEN
        LXI D,FNDMSG       ;FILE ALREADY EXISTS. GO TO CPM
        CALL BDOS
        CALL CRLF
        JMP CPM
NOFILE  MVI C,09H          ;PRINT STRING TO SCREEN
        LXI D,NOMSG        ;NC FILE TRANSFER
        CALL BDOS
```

```
         CALL CRLF
         JMP CPM
ABORT    CALL CRLF
         MVI A,CTRLC          ;SEND XMITING MICRO ABORT ACK
         CALL POUT
         MVI C,09H            ;PRINT STRING TO SCREEN
         LXI D,ABRTMSG        ;XMITING MICRO ABORTED
         CALL BDOS
         CALL CRLF
         JMP GOCPM            ;GO TO CPM
FULLDSK  MVI A,DSKFUL         ;'d'
         CALL POUT            ;TELL XMITING MICRO DISK FULL
         CALL PIN             ;AWAITING CONFIRMATION
         CPI DONE
         JNZ FULLDSK
         MVI C,09H            ;PRINT TO SCREEN
         LXI D,FULLMSG        ;FILE TRANSFER INCOMPLETE, DISK FULL
         CALL BDOS
         CALL CRLF
         JMP CPM
CHECK    CALL STATIN2         ;CHECK STATUS BYTE
         JZ CHECK             ;CONTINUE UNTIL TXRDY IS SET
         RET
STATIN1  IF NORTHSTAR         ;MICRO IS NORTHSTAR
```

```
         IN STATUS
         ANI RXRDY
        ENDIF
        IF APPLE                ;MICRO IS APPLE
         LDA STATUS1
         ANI RXRDY1
        ENDIF
        RET
STATIN2 IF NORTHSTAR            ;MICRO IS NORTHSTAR
         IN STATUS
         ANI TXRDY
        ENDIF
        IF APPLE                ;MICRO IS APPLE
         LDA STATUS1
         ANI TXRDY1
        ENDIF
        RET
PIN     IF NORTHSTAR            ;MICRO IS NORTHSTAR
         IN DATA
        ENDIF
        IF APPLE                ;MICRO IS APPLE
         LDA DATA1
        ENDIF
        RET
```

```
CRLF    MVI A,0DH           ;CARRIAGE RETURN
        CALL OUTPUT
        MVI A,0AH           ;LINE FEED
        CALL OUTPUT
        RET
OUTPUT  PUSH H              ;SAVE THE H,
        PUSH D              ;D,
        PUSH B              ;AND B REGISTERS
        PUSH PSW
        MVI C,CONOUT        ;PRINT TO SCREEN
        MOV E,A             ;PUT THE ACCUMULATOR IN 'E' RGSTR
        CALL BDOS
        POP PSW
        POP B               ;RETURN THE B,
        POP D               ;D,
        POP H               ;AND H REGISTERS
        RET
BDOS    PUSH H              ;SAVE THE H,
        PUSH D              ;D
        PUSH B              ;AND B REGISTERS
        CALL BDOS1          ;EXECUTE
        POP B               ;RETURN THE B,
        POP D               ;D,
        POP H               ;AND H REGISTERS
```

```
        RET
BDOS1   EQU 0005H
EXX     EQU 0D9H                ;FLIP THE REGISTERS
ATTN    EQU 52H                 ;'R'
RXACK   EQU 72H                 ;'r'
DATA    EQU 04H                 ;FOR NORTHSTAR
DATA1   EQU 0E09FH              ;FOR APPLE
TXRDY   EQU 01H                 ;FOR NORTHSTAR
TXRDY1  EQU 02H                 ;FOR APPLE
RXRDY   EQU 02H                 ;FOR NORTHSTAR
RXRDY1  EQU 01H                 ;FOR APPLE
STATUS  EQU 05H                 ;STATUS PORT FOR NORTHSTAR
STATUS1 EQU 0E09EH              ;STATUS PORT FOR APPLE
TXSYM   EQU 12H                 ;DC2 SYMBOL
TXACK   EQU 74H                 ;'t'
RLDTA   EQU 0C3H                ;0CBH MEANS DATA
GOOD    EQU 67H                 ;'g'
BAD     EQU 62H                 ;'b'
DSKFUL  EQU 64H                 ;'d'
DMA     EQU 80H                 ;ADDRESS OF DMA
ENDMA   EQU 01H                 ;LAST LOCATION IN DMA
FCB     EQU 005CH               ;ADDRESS OF FCB
CTRLC   EQU 03H                 ;CONTROL C MEANS GO TO CPM
CPM     EQU 0000H
```

```
GOON    EQU 47H             ;'G' MEANS CONTINUE
DONE    EQU 5AH             ;'Z' MEANS END OF SESSION
QUIT    EQU 14H             ;DC4 SYMBOL MEANS FILE COMPLETE
CONIN   EQU 01H             ;CHECK CONSOLE BUFFER FOR INPUT
CONOUT  EQU 02H             ;OUTPUT CURRENT A REG BYTE TO SCREEN
RIGHTS  DB 'MICROLAN   VERSION 2.0',13,10
        DB 'COPYRIGHT (C) 1985  ROGER D. JASKOT and HAROLD W. HENRY$'
ENTER   DB 'ENTER NAME OF FILE TO BE SENT. IF THE FILE IS ON',13,10
        DB 'A DISK IN THE OTHER DRIVE, ENTER IN THE FORMAT:',13,10,10
        DB '            B:FILENAME.FILETYPE$'
WCHDSK  DB 'Write file to which disk drive?  Enter A for A drive,',13,10
        DB 'B for B drive, or press return for default drive.$'
RXMODE  DB 'IN RECEIVE MODE.$'
FNFDMSG DB 'FILE DOES NOT EXIST, RETURNING TO CPM.$'
TXING1  DB 'TRANSMITTING FILE.$'
HASFILE DB 'RXING MICRO HAS FILE ALREADY, GOING TO CPM.$'
FULMSG  DB 'RXING MICRO DISK FULL.  RETURNING TO CPM.$'
WELCUM  DB 'WELCOME!',13,10,10
        DB 'YOU ARE NOW ENTERING THE TRANSFER ZONE!$'
INSTRC  DB 'Enter an S for transmit mode, an R for receive mode,',13,10
        DB 'or an X to exit.$'
FNDMSG  DB 'FILE ALREADY EXISTS. RETURNING TO CPM.$'
EOFMSG  DB 'FILE TRANSMISSION COMPLETED.$'
NOMSG   DB 'NO FILE TRANSFER. RETURNING TO CPM.$'
```

```
ABRTMSG DB 'XMITING MICRO ABORTED FILE TRANSFER.',13,10
        DB 'PLEASE ERASE FILENAME FROM YOUR DIRECTORY.$'
FULLMSG DB 'DISK FULL. FILE TRANSFER INCOMPLETE.$'
ERMSG   DB 'ENTER FILENAME AGAIN. END WITH <CR>$'
RXING1  DB 'CONNECTION MADE.$'
SWAP    DB EXX                    ;SAVE THE REGISTERS USING EXX
        RET
CONBUF  DB 16                     ;BUFFER FOR FILENAME
        DB 00
        DS 16
```

# APPENDIX D

## CP/M - 86

```
     CSEG
     ORG  0100H
     MOV DX,OFFSET BAUDMSG     ;BAUDRATE HEADER
     MOV CL,09H                ;PRINT SAME
     INT 0E0H
     MOV CL,01H                ;GET KEYBOARD INPUT
     INT 0E0H
     SUB AL,31H                ;CONVERT TO TABLE OFFSET
     CMP AL,05H
     JBE SETB1
     JMP ERROR1
SETB1:   MOV BX,OFFSET TABL
     ADD AL,AL
     MOV AH,0
     ADD BX,AX
     MOV DX,[BX]
     MOV BX,OFFSET BAUD
     MOV [BX],DX
     MOV DX,03FBH              ;LINE CONTROL
     MOV AL,83H                ;DLAB=1
     OUT DX,AL
```

```
       MOV DX,03F8H             ;BAUDATE DIVISOR
       MOV BX,OFFSET BAUD
       MOV AX,[BX]
       OUT DX,AX
       MOV DX,03FBH             ;CONTROL
       MOV AL,03H
       OUT DX,AL                ;RESET DLAB
   INIT: CALL CRLF
       MOV   CL,09H             ;PRINT STRING TO SCREEN
       MOV   DX,OFFSET RIGHTS   ;COPYRIGHTS
       CALL BDOS
       CALL CRLF
       MOV   CL,09H             ;PRINT STRING TO SCREEN
       MOV   DX,OFFSET WELCUM   ;WELCOME MSG
       CALL BDOS
       CALL CRLF
       CALL CRLF
       MOV   CL,09H             ;PRINT STRING TO SCREEN
       MOV   DX,OFFSET INSTRC   ;SEND, RECEIVE, OR QUIT?
       CALL BDOS
       CALL CRLF
       CALL CRLF
   HOLDING: MOV   CL,06H        ;CHECK FOR CONSOLE INPUT
       MOV   DL,0FFH            ;LOOKING FOR INPUT
```

```
    CALL BDOS
    AND  AL,0DFH              ;ENSURE LETTER IS A CAPITAL
    CMP  AL,53H               ;IS IT AN 'S'?
    JNE G1 ! JMP MASTER       ;IF SO, START FILE TRANSFER
G1: CMP  AL,52H               ;IS IT AN 'R'?
    JNE G2 ! JMP SLAVE        ;IF SO, PREPARE TO RECEIVE FILE
G2: CMP  AL,58H               ;IS IT AN 'X'?
    JNE G3 ! JMP CPM          ;IF SO, GO TO CPM
G3: JMP  HOLDING              ;REPEAT UNTIL INPUT FOUND
MASTER: MOV   CL,09H          ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET ENTER     ;ENTER FILENAME
    CALL BDOS
    CALL CRLF
    CALL CRLF
FILLUP: MOV   BX,FCB          ;ADDRESS OF FCB
    MOV [BX],BYTE PTR 00H
    INC  BX
    MOV   CL,0BH              ;11 SPACES
FILLUP1: MOV [BX],BYTE PTR 20H ;FILL MEMORY ADDRESS WITH SPACES
    INC  BX                   ;MOVE PTR TO NEXT ADDRESS
    DEC  CL                   ;DECREMENT COUNTER
    JNZ  FILLUP1              ;REPEAT UNTIL DONE
    MOV   CL,13H              ;TOTAL OF 20 SPACES
FILLUP2: MOV [BX],BYTE PTR 00H ;FILL REST OF ADDRESS WITH 0'S
```

```
    INC  BX                      ;MOVE PTR TO NEXT ADDRESS
    DEC  CL                      ;DECREMENT COUNTER
    JNZ  FILLUP2                 ;REPEAT UNTIL DONE
HOLD1: MOV   CL,0AH              ;READ CONSOLE BUFFER
    MOV   DX,OFFSET CONBUF       ;ADDRESS OF FIRST LETTER OF FILENAME
    CALL BDOS
    MOV   BX,OFFSET CONBUF       ;ADDRESS OF CONSOLE BUFFER
    MOV   DX,FCB+1               ;FCB ADDRESS
    INC  BX
    MOV   CH,[BX]                ;STORE COUNT IN BX REGISTER
    MOV   AL,[BX]                ;MOVE COUNT TO ACCUMULATOR
    OR   AL,AL                   ;IS THERE AN INPUT?
    JNE G4 ! JMP ERROR           ;TRY AGAIN
G4: INC  BX
FLUP: MOV   AL,[BX]
    CMP  AL,3AH                  ;IS CHARACTER A ':'?
    JNE G5 ! JMP DSKSEL          ;IF SO, GO TO DISK SELECT
G5: CMP  AL,2EH                  ;IS IT A '.'?
    JNE G6 ! JMP FIXIT           ;IF SO, SKIP TO FILETYPE
G6: CMP  AL,40H                  ;CHECK FOR LETTER
    JNC G7 ! JMP DONTFIX         ;SKIP NEXT STEP IF NOT LETTER
G7: AND  AL,0DFH                 ;ENSURE LETTER IS A CAPITAL
DONTFIX: XCHG BX,DX ! MOV [BX],AL ! XCHG BX,DX ;STORE LETTER IN FCB
    INC  DX
```

```
    INC  BX
    DEC  CH
    JNZ  FLUP                    ;REPEAT UNTIL END
LISN1: MOV   AL,ATTN             ;LETTER 'R'
    CALL POUT1
    MOV   CL,03H                 ;LISTEN 3 TIMES
LISN: CALL PIN
    CMP  AL,RXACK                ;IS IT AN 'r'?
    JNE G8 ! JMP XMIT            ;IF SO, THEN XMIT
G8: DEC  CL                      ;OTHERWISE DECREMENT CTR
    JNZ  LISN                    ;LISTEN UNTIL CTR IS ZERO
    JMP  LISN1                   ;THEN TRY AGAIN
XMIT: CALL CRLF
    CALL CRLF
    MOV   CL,09H                 ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET RXING1       ;'r' WAS RECEIVED
    CALL BDOS
    CALL CRLF
XMIT1: MOV   AL,TXSYM            ;DC2 SYMBOL FOR SYNC AT START
    CALL POUT1                   ;OF 128 BYTE BLOCK
    MOV   CL,08AH                ;LISTEN 138 TIMES
LITTLET: CALL PIN
    CMP  AL,TXACK                ;WAS 't' RECEIVED?
    JNE G9 ! JMP TXFCE           ;IF SO, XMIT FILE CTRL BLK
```

```
G9: DEC  CL                    ;OTHERWISE KEEP LISTENING
    JNZ  LITTLET               ;UNTIL CTR IS ZERO,
    JMP  XMIT1                 ;THEN SEND DC2 SYNC AGAIN
TXFCB: CALL CRLF               ;SEND FILENAME TO RXING MICRO
    CALL OPENIT                ;SEE IF FILE EXISTS.  IF SO, OPEN IT
TXFCB1: MOV   CH,00H           ;INITIALIZE CHECKSUM REGISTER
    MOV   BX,FCB               ;SET PTR TO 1ST LETTER IN FILENAME
FCBLUP: MOV  AL,CH             ;PERFORM CHECKSUM OPERATION
    INC  BX                    ;MOVE PTR TO NEXT BYTE
    XOR  AL,[BX]               ;BY XORING CURRENT BYTE
    MOV  CH,AL                 ;WITH B REGISTER
    MOV  AL,[BX]               ;PUT CURRENT BYTE IN ACCUM
    CALL POUT1                 ;SEND CURRENT BYTE
    CMP  AL,0H                 ;CHECK FOR END OF FILENAME
    JNE G10 ! JMP FCBCK        ;IF END, GO TO CHECKSUM LOOP
G10: JMP  FCBLUP               ;IF NOT, REPEAT FCB LOOP
FCBCK: MOV   CL,20H            ;LOOP 32 TIMES
FCBCK1: CALL PIN               ;FOR SYNC WITH SLAVE
    CMP  AL,RXACK              ;IS IT AN 'r'
    JNZ  FCBCK1                ;IF NOT, LISTEN AGAIN.  IF SO,
    MOV  AL,CH                 ;PUT CHECKSUM IN ACCUM
    CALL POUT                  ;SEND CHECKSUM
    PUSH CX                    ;SAVE CHECKSUM
    MOV   AL,0H                ;CLEAR ACCUM
```

```
    MOV   CH,80H              ;LISTEN 100 TIMES
FCBTMCT: CALL PIN             ;READ MAIL
    CMP  AL,BAD               ;DID IT CHECK BAD?
    JNE G11 ! JMP RSNDFCB     ;IF SO, SEND FCB AGAIN
G11: CMP  AL,GOOD             ;DID IT CHECK GOOD?
    JNE G12 ! JMP WAITFIL     ;IF SO, GO TO NEXT ROUTINE
G12: DEC  CH                  ;IF NOT, DECREMENT CTR, AND
    JNZ  FCBTMOT              ;IF NOT 0, LISTEN AGAIN
    POP  CX                   ;CLEAR STACK
    DEC  CL                   ;IF SO, DECREMENT CL
    JNZ  FCBCK1               ;AND REPEAT UNTIL CL=0
    JMP  TXFCB                ;IF 0, ASSUME PROBLEM AND SEND AGAIN
WAITFIL: POP  CX              ;CLEAR STACK
    MOV   CX,07FFH            ;COUNT LOOP APPX 2K
WAIT1: CALL STATIN1           ;ANY 'MAIL'?
    JZ   WAIT1                ;IF NOT, CHECK AGAIN
    DEC  CX                   ;IF SO, DECREMENT CTR
    JNE G13 ! JMP GOCPM       ;AND, IF 0, QUIT
G13: CALL PIN                 ;OTHERWISE READ 'MAIL'
    CMP  AL,QUIT              ;DOES RXING MICRO ALREADY HAVE FILE?
    JNE G14 ! JMP GOCPM1      ;IF SO, GO TO CPM
G14: CMP  AL,GOON             ;IS IT THE GO ON SIGNAL 'G'
    JNZ  WAITFIL              ;IF NOT, LISTEN AGAIN.  ALLOW RXING
                              ;MICRO TO CATCH UP
```

```
    CALL CRLF
TXDATA:                         ;SEND THE FILE
    MOV   CL,09H                ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET TXING1      ;SAYS FILE BEING SENT
    CALL BDOS
    CALL CRLF
RDSEQ: CALL READSEQ             ;READ FIRST 128 BYTE BLOCK
SEND: CALL CHECK                ;AND SEND TO RXING MICRO
    MOV   AL,TXSYM              ;DC2 SYMBOL FOR SYNC AT START OF DATA
    CALL POUT1
    MOV   CL,0FH                ;LISTEN 15 TIMES
LITLET2: CALL PIN
    CMP  AL,TXACK               ;IS IT A 't'?
    JNE G15 ! JMP SLUP2         ;IF SO, READY TO SEND DATA
G15: CMP  AL,DSKFUL             ;IS RXING MICRO'S DISK FULL?
    JNE G16 ! JMP FULDISK       ;IF SO, QUIT
G16: DEC  CL                    ;IF NOT, DECREMENT CTR
    JNZ  LITLET2                ;LISTEN AGAIN, UNLESS CTR IS 0,
    JMP  SEND                   ;THEN TRY TO SYNC AGAIN
SLUP2: MOV AL,PLDTA             ;0CBH MEANS TIME FOR DATA
    CALL POUT1
    MOV CX,07FFH                ;WAIT LOOP APPX 2K
SLUP3: CALL PIN
    CMP AL, PLDTA               ;IS IT AN ECHO?
```

```
    JZ SLUP1                    ;IF SO, SEND DATA
    DEC CX                      ;IF NOT, DECREMENT COUNTER
    JNZ SLUP3                   ;REPEAT UNTIL ZERO
    JMP SLUP2
SLUP1: MOV   BX,DMA             ;POINTER TO 1ST INFO BYTE
    MOV   CH,00H                ;INITIALIZE CHECKSUM LOCATION
SLOOP: MOV  AL,CH               ;PERFORM CHECKSUM
    XOR  AL,[BX]                ;XOR DATA WITH CH REGISTER
    MOV  CH,AL
    MOV  AL,[BX]                ;PUT BYTE IN ACCUMULATOR
    CALL POUT                   ;DATA IS TRANSFERRED
    INC  BX                     ;MOVE PTR TO NEXT BYTE
    MOV  AL,BH                  ;MOVE BH REG TO ACCUM
    CMP  AL,ENDMA               ;END DMA, CHECK FOR LAST BYTE
    JNZ  SLOOP                  ;IF NOT, SEND NEXT BYTE.  OTHERWISE
CKC: MOV  AL,CH                 ;PUT CHECKSUM IN ACCUMULATOR
    CALL POUT                   ;AND SEND TO RXING MICRO
CLCTMCT: MOV   CH,01AH          ;LISTEN 26 TIMES
CKCT1: CALL STATIN1             ;CHECK INPUT BUFFER
    JZ   CKCT1                  ;IF NOTHING, TRY AGAIN
    CALL PIN                    ;READ MAIL
    CMP  AL,BAD                 ;IS CHECK BAD?
    JNE G17 ! JMP RESEND        ;IF SO, SEND BLOCK AGAIN
G17: CMP  AL,GOOD               ;IS CHECK GOOD?
```

```
    JNE G18 ! JMP RDSQRPT        ;IF SO, READ NEXT BLOCK
G18: DEC  CH                     ;DECREMENT COUNTER
    JNZ  CRCT1                   ;IF NOT TIMED OUT,LISTEN AGAIN
    JMP  SEND                    ;IF TIMED OUT, ASSUME PROBLEM.
                                 ;SEND BLOCK AGAIN
DSKSEL: MOV   BX,OFFSET CONBUF+2 ;ADDRESS OF DISK SEL ENTRY
    MOV  AL,[BX]                 ;PUT DISK SEL IN ACCUM
    AND  AL,0DFH                 ;ENSURE LETTER IS CAPITAL
    CMP  AL,'A'                  ;IS LETTER AN 'A'?
    JZ ADISK                     ;IF SO, SET FOR A DRIVE.
    CMP  AL,'B'                  ;IS LETTER A 'B'?
    JZ BDISK                     ;IF SO, SET FOR B DRIVE.
    CMP AL,'C'                   ;IS LETTER A 'C'?
    JZ CDISK                     ;IF SO, SET FOR C DRIVE.
    JMP DSKSEL1                  ;IF NEITHER, RETURN TO FILENAME LOOP.
ADISK: MOV   DI,FCB              ;SET PTR TO DRIVE BYTE.
    MOV [DI],BYTE PTR 01H        ;SET FCB FOR A DRIVE.
    JMP  DSKSEL1                 ;RETURN TO FILENAME LOOP.
BDISK: MOV   DI,FCB              ;SET PTR TO DRIVE BYTE.
    MOV [DI],BYTE PTR 02H        ;SET FCB FOR B DRIVE.
    JMP  DSKSEL1                 ;RETURN TO FILENAME LOOP.
CDISK: MOV   DI,FCB              ;SET PTR TO DRIVE BYTE.
    MOV [DI],BYTE PTR 03H        ;SET FCB FOR C DRIVE.
DSKSEL1: INC  BX                 ;MOVE BUFFER POINTER TO FILENAME.
```

```
    INC BX
    MOV   DX,FCB+1          ;FCB FILENAME ADDRESS.
    JMP  FLUP               ;RETURN TO FILENAME LOOP.
RESEND: MOV   CL,CONOUT     ;PRINT TO SCREEN
    MOV   DL,BAD            ;A 'b' IF CHECKSUM WAS BAD
    CALL BDOS
    JMP  SEND               ;AND SEND BLOCK AGAIN
RSNDFCB: POP  CX            ;RECALL CHECKSUM
    MOV  AL,CH              ;PUT CHECKSUM IN ACCUM
    ADD  AL,3               ;ADD 3 TO OFFSET
    CALL POUT1              ;SEND BYTE
RSNDF: CALL PIN
    CMP  AL,RXACK           ;IS IT AN 'r'
    JNZ  RSNDF              ;IF NOT LISTEN AGAIN
RSNDFC1: CALL PIN           ;READ MAILBOX
    CMP  AL,TXACK           ;SYNC WITH RXING MICRO
    JNZ  RSNDFC1            ;REPEAT UNTIL TXACK RECEIVED
    JMP  TXFCB1             ;IF SO, RESEND FCB
FIXIT: MOV   DX,FCB+9       ;MOVE POINTER TO FILETYPE AREA
    INC  BX                 ;MOVE PTR TO FIRST LETTER OF FILTYPE
    JMP  FLUP
ERROR: MOV   CL,09H         ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET ERMSG   ;ERROR MESSAGE
    CALL BDOS
```

```
    CALL CRLF
    JMP  HOLD1                ;LOOK FOR INPUT AGAIN
POUT1: LAHF ! PUSH AX         ;SEND THE DATA
                              ;FIRST, SAVE THE CURRENT BYTE
    MOV   CL,06H              ;CHECK FOR CONSOLE INPUT
    MOV   DL,0FFH             ;LOOKING FOR INPUT
    CALL BDOS
    CMP  AL,CTRLC             ;IS THERE A CONTROL C?
    JNE G22 ! JMP STOPS
G22: CALL CHECK               ;PERFORM CHECK
    POP AX ! SAHF             ;AND RECALL BYTE
    MOV DX,DATA
    OUT DX,AL
    RET
OPENIT: MOV   CL,0FH          ;OPEN FILE CODE
    MOV   DX,FCB              ;FILE CTRL BLOCK ADDRESS IN DX REG PR
    CALL BDOS
    CMP  AL,0FFH              ;FF = FILE NOT FOUND
    JNE G23 ! JMP FNFOUND     ;IF FILE NOT FOUND
G23: RET                      ;OTHERWISE, RET TO TX DATA
CLOSIT: MOV   CL,10H          ;CLOSE FILE CODE
    MOV   DX,FCB              ;FILE CTRL BLOCK ADDRESS IN DX REG PR
    CALL BDOS
CLOSIT1: MOV   AL,DONE        ;END OF SESSION MSG 'Z'
```

```
    CALL POUT1                  ;SEND TO RXING MICRO
    MOV   AL,0H                 ;CLEAR ACCUM
    CALL PIN                    ;CHECK REPLY
    CMP  AL,DONE                ;DOES RXING MICRO AGREE?
    JNZ  CLOSIT1                ;IF NOT, REPEAT
    JMP  GOCPM                  ;IF SO, GO TO CPM
READSEQ: PUSH CX
    PUSH DX
    MOV   CL,14H                ;READ SEQUENTIAL CODE
    MOV   DX,FCB                ;FILE CTRL BLOCK ADDRESS IN DX REG PR
    CALL BDOS
    POP DX
    POP CX
    CMP  AL,0                   ;0 MEANS SUCCESSFUL READ
    JZ G24 ! JMP EOFILE         ;IF NOT 0, ASSUME FINISHED WITH FILE
G24: RET
RDSQRPT: MOV   CL,CONCUT        ;PRINT TO SCREEN
    MOV   DL,02AH               ;'*' SO USER KNOWS BLK WAS SENT
    CALL BDOS
    JMP  RDSEQ                  ;TO READ NEXT 128 BYTE BLK
FNFOUND: MOV   AL,QUIT          ;TELL RXING MICRO NO FILE FOUND
    CALL POUT1
    MOV   CL,09H                ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET FNFDMSG     ;FILE NOT FOUND MSG
```

```
    CALL BDOS
    CALL CRLF
    JMP  GOCPM                 ;AND GO TO CPM
EOFILE: MOV   AL,TXSYM        ;DC2 SYMBOL FOR SYNC WITH RXING MICRO
    CALL POUT1
    MOV   CL,0FH               ;LISTEN 15 TIMES
LITLET3: CALL STATIN1          ;CHECK FOR MAIL
    JZ   LITLET3               ;IF NONE, CHECK AGAIN
    CALL PIN                   ;READ MAIL
    CMP  AL,TXACK              ;IS IT A 't'?
    JNE G25 ! JMP EOFIL1       ;IF SO, CONTINUE
G25: DEC  CL                   ;IF NOT, DECREMENT COUNTER
    JNZ  LITLET3               ;AND LISTEN AGAIN, UNLESS COUNTER IS
    JMP  EOFILE                ;0.  THEN TRY AGAIN
EOFIL1: MOV   AL,QUIT          ;DC4 SYMBOL.  TELLS RXING MICRO THAT
    CALL POUT1                 ;THE FILE IS DONE
    CALL PIN                   ;LISTEN FOR REPLY
    CMP  AL,QUIT               ;DOES RXING MICRO ACKNOWLEDGE?
    JNZ  EOFIL1                ;IF NOT, TRY AGAIN
    CALL CRLF
    MOV   CL,09H               ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET EOFMSG     ;IF SO, TELL USER FILE IS DONE
    CALL BDOS
    CALL CRLF
```

```
    JMP  CLOSIT                ;AND CLOSE THE FILE
STOPS: MOV   AL,CTRLC          ;SEND CTRLC TO RXING MICRO
    MOV DX,DATA
    OUT  DX,AL
    MOV AL,0H                  ;CLEAR ACCUM
    CALL PIN                   ;FROM RXING MICRO
    CMP  AL,CTRLC              ;ACK FROM RXING MICRO
    JNZ  STOPS                 ;REPEAT UNTIL ACK
    JMP  GOCPM
FULDISK: MOV   AL,DONE         ;LETTER 'Z' TO ACKNOWLEDGE
    CALL POUT1                 ;SEND BYTE
    MOV   CL,09H               ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET FULMSG     ;SAYS RXER'S DISK FULL
    CALL BDOS
    CALL CRLF
    JMP  GOCPM
GOCPM: MOV   AL,0H             ;RESET THE ACCUMULATOR AND
    MOV DX,DATA
    OUT  DX,AL                 ;CLEAR OUTPUT BUFFER
    CALL CRLF
    CALL PIN
    JMP  CPM                   ;AND GO TO CPM
GOCPM1: MOV   CL,09H           ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET HASFILE    ;RXING MICRO HAS FILE ALREADY
```

```
    CALL BDOS
    JMP  GOCPM
SLAVE: MOV   CL,09H          ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET WCHDSK   ;SELECT DISK DRIVE
    CALL BDOS
    CALL CRLF
    CALL CRLF
DRVSEL: MOV   CL,06H         ;CHECK FOR CONSOLE INPUT
    MOV   DL,0FFH            ;LOOKING FOR INPUT
    CALL BDOS
    CMP  AL,0DH              ;IS IT A <CR>?
    JNE G26 ! JMP CONT       ;IF SO, ENTER RECEIVE MODE
G26: AND  AL,0DFH            ;ENSURE LETTER IS A CAPITAL
    CMP  AL,'A'              ;IS IT AN 'A'?
    JZ G27 ! JMP DISKB       ;SKIP TO B IF NOT 'A'
G27: MOV   BX,FCB            ;ADDRESS OF DISK DRIVE BYTE
    MOV [BX],BYTE PTR 01H    ;SET BYTE TO A DISK DRIVE
    JMP  CONT                ;THEN CONTINUE
DISKB: CMP  AL,'B'           ;IS IT A 'B'?
    JNZ  DISKC               ;IF NOT, SKIP TO C
    MOV  BX,FCB              ;ADDRESS OF DISK DRIVE BYTE
    MOV [BX],BYTE PTR 02H    ;SET BYTE TO B DISK DRIVE
    JMP CONT                 ;THEN CONTINUE
DISKC: CMP AL,'C'            ;IS IT A 'C'?
```

```
    JNZ DRVSEL                  ;IF NOT, LISTEN AGAIN
    MOV  BX,FCB                 ;ADDRESS OF DISK DRIVE BYTE
    MOV [BX],BYTE PTR 03H       ;SET BYTE TO C DISK DRIVE
CONT: MOV   CL,09H              ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET RXMODE      ;IN RECEIVE MODE
    CALL BDOS
    CALL CRLF
SLAVE1: MOV   AL,00H            ;RESET ACCUMULATOR
    CALL PIN                    ;LISTENING FOR AN 'R'
    CMP  AL,ATTN                ;'R'
    JNZ  SLAVE1                 ;IF 'R' RX'D, CONTINUE. IF NOT
                                ;LISTEN AGAIN
    CALL CRLF
    MOV   CL,09H                ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET RXING1      ;CONNECTION MADE
    CALL BDOS
    CALL CRLF
    MOV   AL,RXACK              ;'r'
    CALL POUT                   ;SEND AN 'r' TO XMITING MICRO
LISTEN: CALL PIN1               ;LISTENING FOR A 'DC2'
    CMP  AL,TXSYM               ;'DC2'
    JNZ  LISTEN                 ;IF 'DC2' RX'D, CONTINUE. IF NOT,
                                ;LISTEN AGAIN
    CALL CRLF
```

```
RXFCB: MOV   BX,FCB+1          ;ADDRESS OF FCB MEM LOC INTO BX REG PR
    MOV   CL,1EH               ;COUNTER FOR FCB'S 31 SPACES
RSTFCB: MOV [BX],BYTE PTR 00H  ;FILL FCB WITH 0'S
    INC  BX                    ;MOVE PTR TO NEXT MEMORY ADDRESS IN FCB
    DEC  CL                    ;DECREMENT COUNTER
    MOV  AL,CL
    CMP  AL,0
    JNZ  RSTFCB                ;IF COUNTER = 0, CONT. IF NOT,
                               ;PUT ANOTHER 0 IN FCB
    MOV   CH,00H               ;INITIALIZE CHECKSUM
    MOV   BX,FCB+1             ;LOAD 2ND ADDRESS OF FCB IN
    MOV   AL,TXACK             ;'t'
    CALL POUT                  ;SEND 't' TO XMITING MICRO FOR SYNC
    CALL PIN1                  ;CLEAR THE ACCUMULATOR
RST1: CALL STATIN1             ;CHECKING FOR INPUT
    JZ   RST1
RST2: CALL PIN1                ;FILE NAME DATA
    CMP  AL,QUIT               ;IS DATA A 'QUIT'?
    JNE G28 ! JMP NOFILE       ;FILE DID NOT EXIST
G28: CMP  AL,0H                ;CHECK IF FILENAME COMPLETELY SENT
    JNE G29 ! JMP FCBCRC       ;IF FILENAME RX'D, GO TO CHECKSUM
G29: CMP  AL,TXSYM             ;CHECK IF DATA IS VALID
    JZ   RST2                  ;IF DATA IS NOT FILENAME,
    MOV  [BX],AL               ;PUT FILENAME IN FCB
```

```
    CALL OUTPUT                 ;PRINT FILENAME TO SCREEN
    MOV  AL,CH
    XOR  AL,[BX]                ;CALCULATE CHECKSUM
    MOV  CH,AL
    INC  BX                     ;MOVE PTR TO NEXT FCB ADDRESS
    JMP  RST1
FCBCRC: MOV   AL,RXACK          ;'r'
    CALL POUT                   ;SYNC DATA WITH XMITING MICRO
FCBCRC1: CALL STATIN1           ;CHECKING FOR INPUT
    JZ   FCBCRC1
    CALL CRLF
    CALL PIN                    ;CHECKSUM DATA
    CMP  AL,CH                  ;COMPARE CHECKSUM
    JNE G30 ! JMP STRTFIL       ;CHECKSUM MATCHED
G30: ADD  AL,3                  ;ADD 3 TO THE CHECKSUM
    MOV  CL,AL                  ;STORE IN REGISTER
    MOV   AL,BAD                ;CHECKSUM DID NOT MATCH
    CALL POUT                   ;TELL XMITING MICRO
CLEAR: CALL PIN
    CMP  AL,CL                  ;XMITING MICRO STOPPED SENDING CHKSUM?
    JNZ  CLEAR                  ;IF NOT,LISTEN AGAIN
    MOV   AL,RXACK              ;SYNC WITH XMITING MICRO
    CALL POUT
    JMP  RXFCB                  ;TRY AGAIN
```

```
STRTFIL: MOV   AL,GOOD          ;READY TO CHECK IF FILE ALREADY PRESENT
    CALL POUT
    CALL OPNFILE                ;CHECK IF FILE EXISTS
    CALL MAKEFIL                ;CREATE NEW FILE
RXD1: MOV   CH,00H              ;INITIALIZE CHECKSUM
    MOV   BX,DMA                ;LOAD ADDRESS OF DMA MEM LOC TO
                                ;BX REGISTER PAIR
    MOV   CL,81H                ;INITIALIZE COUNTER WITH SIZE OF DMA
RXDS: CALL STATIN1
    JZ RXDS
RXD2: CALL PIN1                 ;SYNC WITH XMITING MICRO
    CMP  AL,TXSYM               ;COMPARE WITH 'DC4'
    JNZ  RXD2
    MOV   AL,TXACK              ;'t'
    CALL POUT                   ;IN SYNC WITH XMITING MICRO
RXDS1: CALL STATIN1             ;CHECKING FOR INPUT
    JZ   RXDS1
RXYET: CALL PIN1
    CMP AL,RLDTA                ;IS IT 0CBH?
    JZ RXYET1                   ;IF SO, GO TO RECEIVE DATA
    CMP AL,QUIT                 ;IS IT 'DC4' FOR QUIT?
    JNZ RXYET4                  ;IF NOT, LISTEN AGAIN; OTHERWISE,
    JMP CLSFILE                 ;IF SO, CLOSE FILE
RXYET4: JMP RXDS1               ;LISTEN AGAIN
```

```
RXYET1: MOV AL,RLDTA            ;ACK REAL DATA COMING
    CALL POUT
    MOV AL,00H                  ;CLEAR ACCUM
RXYET2: CALL STATIN1            ;CHECKING FOR INPUT
    JZ RXYET2
RXYET3: CALL PIN                ;READ DATA
    CMP AL,RLDTA                ;IS IT STILL RLDTA?
    JZ RXYET3
RXD3: DEC  CL                   ;DECREMENT COUNTER
    JNE G32 ! JMP RXCRC         ;CHECKSUM RX'D
G32: MOV [BX],AL                ;PUT THE DATA IN MEMORY
    MOV  AL,CH
    XOR  AL,[BX]                ;CALCULATE CHECKSUM
    MOV  CH,AL
    INC  BX                     ;MOVE PTR TO NEXT DMA ADDRESS
RXD4: CALL STATIN1              ;CHECK FOR INPUT
    JZ RXD4                     ;LOOP UNTIL INPUT
    CALL PIN
    JMP  RXD3
RXCRC: MOV  AL,AL               ;ENSURE CH IS COMPARED TO A
    CMP  AL,CH                  ;COMPARE WITH CHECKSUM
    JNE G33 ! JMP WRITFIL       ;128 BYTE BLOCK SENT
G33: MOV   AL,BAD               ;CHECKSUM DID NOT MATCH
    CALL POUT                   ;NOTIFY XMITING MICRO
```

```
    JMP  RXD2                   ;SEND 128 BYTE BLOCK AGAIN
POUT: LAHF ! PUSH AX            ;SAVE THE DATA
    CALL CHECK
    POP AX ! SAHF               ;RETURN THE DATA
    MOV DX,DATA
    OUT DX,AL                   ;SEND DATA
    RET
PIN1: MOV DX,DATA
    IN AL,DX
    CMP  AL,CTRLC               ;DID XMITING MICRO ABORT?
    JNE G34 ! JMP ABORT         ;IF SO, ABORT
G34: RET
WRITFIL: MOV   AL,GOOD          ;XMIT THAT THE CHECKSUM IS CORRECT
    CALL POUT
    CALL WRITSEQ                ;START WRITING FILE TO DISK
    MOV   CL,CONOUT             ;PRINT TO SCREEN
    MOV   DL,02AH               ;'*' TO PRINT TO SCREEN
    CALL BDOS
    JMP  RXD1
OPNFILE: MOV   CL,0FH           ;OPEN FILE CODE
    MOV   DX,FCB                ;FCB ADDRESS IN DX RGSTR PAIR
    CALL BDOS
    CMP  AL,0FFH                ;FF = FILE NOT FOUND
    JZ G35 ! JMP FILFND         ;FILE EXISTS
```

```
G35: RET
CLSFILE: MOV   AL,QUIT          ;'DC4'
    CALL POUT                   ;AGREE END OF FILE
    MOV   CL,10H                ;CLOSE FILE CODE
    MOV   DX,FCB                ;FCB ADDRESS IN DX RGSTR PAIR
    CALL BDOS
    CALL CRLF
    MOV   DX,OFFSET EOFMSG      ;FILE TRANSMISSION COMPLETED
    MOV   CL,09H                ;PRINT STRING TO SCREEN
    CALL BDOS
    CALL CRLF
    MOV   AL,0H                 ;CLEAR THE ACCUMULATOR
CLSFIL1: CALL PIN1              ;LOOKING FOR END OF SESSION MSG
    CMP  AL,DONE                ;'Z' = END OF SESSION
    JNZ  CLSFIL1
    MOV   AL,DONE               ;END OF SESSION MESSAGE
    CALL POUT                   ;CONFIRM RECEPTION OF E-O-SESSION MSG
    JMP  CPM
MAKEFIL: MOV   CL,16H           ;MAKE NEW FILE CODE
    MOV   DX,FCB                ;FCB ADDRESS IN DX RGSTR PAIR
    CALL BDOS
    MOV   AL,GOON               ;CONTINUE MESSAGE
    CALL POUT
    RET                         ;RETURN TO RX FIRST 128 BYTE BLOCK
```

```
WRITSEQ: PUSH CX
    PUSH DX
    MOV   CL,15H              ;WRITE THE FILE TO THE DISK
    MOV   DX,FCB              ;FCB IN DX RGSTR PAIR
    CALL BDOS
    POP DX
    POP CX
    OR    AL,AL               ;CHECK IF DISK IS FULL
    JZ G36 ! JMP FULLDSK      ;IF SO, JUMP TO FULLDSK
G36: RET
FILFND: MOV   AL,QUIT         ;TELL XMITING MICRO, FILE FOUND
    CALL POUT
    MOV   CL,09H              ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET FNDMSG    ;FILE ALREADY EXISTS. GO TO CPM
    CALL BDOS
    CALL CRLF
    JMP  CPM
NOFILE: MOV   CL,09H          ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET NCMSG     ;NC FILE TRANSFER
    CALL BDOS
    CALL CRLF
    JMP  CPM
ABORT: CALL CRLF
    MOV   AL,CTRLC            ;SEND XMITING MICRO ABORT ACK
```

```
    CALL POUT
    MOV   CL,09H             ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET AERTMSG  ;XMITING MICRO ABORTED
    CALL BDOS
    CALL CRLF
    JMP  GOCPM               ;GO TO CPM
FULLDSK: MOV   AL,DSKFUL     ;'D'
    CALL POUT                ;TELL XMITING MICRO DISK FULL
    CALL PIN                 ;AWAITING CONFIRMATION
    CMP  AL,DONE
    JNZ  FULLDSK
    MOV   CL,09H             ;PRINT TO SCREEN
    MOV   DX,OFFSET FULLMSG  ;FILE TRANSFER INCOMPLETE, DISK FULL
    CALL BDOS
    CALL CRLF
    JMP  CPM
CHECK: CALL STATIN2          ;CHECK STATUS BYTE
    JZ   CHECK               ;CONTINUE UNTIL TXRDY IS SET
    RET
STATIN1: MOV DX,STATUS
    IN AL,DX
    AND  AL,RXRDY
    RET
STATIN2: MOV DX,STATUS
```

```
    IN AL,DX
    AND  AL,TXRDY
    RET
PIN: MOV DX,DATA
    IN AL,DX
    RET
CRLF: PUSH AX
    MOV   AL,0DH              ;CARRIAGE RETURN
    CALL OUTPUT
    MOV   AL,0AH              ;LINE FEED
    CALL OUTPUT
    POP AX
    RET
OUTPUT: PUSH ES               ;SAVE THE ES
    PUSH BX                   ;BX,
    PUSH DX                   ;DX,
    PUSH CX                   ;AND CX REGISTERS
    LAHF ! PUSH AX
    MOV  CL,CONOUT            ;PRINT TO SCREEN
    MOV  DL,AL                ;PUT THE ACCUMULATOR IN 'DL' RGSTR
    CALL BDOS
    POP  AX ! SAHF
    POP  CX                   ;RETURN THE CX,
    POP  DX                   ;DX,
```

```
    POP  BX                         ;BX
    POP  ES                         ;AND ES REGISTERS
    RET
BDOS: PUSH ES                       ;SAVE THE ES,
    PUSH BX                         ;BX,
    PUSH DX                         ;DX,
    PUSH CX                         ;AND CX REGISTERS
    INT 0E0H                        ;EXECUTE
    POP  CX                         ;RETURN THE CX,
    POP  DX                         ;DX,
    POP  BX                         ;BX
    POP ES                          ;AND ES REGISTERS
    RET
C?M: MOV DL,00H                     ;RETURN TO OPERATING SYSTEM
    MOV CL,00H
    JMP BDOS
ERROR1: MOV DX,OFFSET ERR1          ;BAUDRATE ERROR
    MOV CL,09H                      ;PRINT STRING TO SCREEN
    INT 0E0H
    MOV CL,0
    MOV DL,0
    INT 0E0H
ATTN      EQU  52H                  ;'R'
RXACK     EQU  72H                  ;'r'
```

```
DATA      EQU  03F8H
TXRDY     EQU  20H
RXRDY     EQU  01H
STATUS    EQU  03FDH
TXSYM     EQU  12H                ;DC2 SYMBOL
TXACK     EQU  74H                ;'t'
RLDTA     EQU   0CBH
GOOD      EQU  67H                ;'g'
BAD       EQU  62H                ;'b'
DSKFUL    EQU  64H                ;'d'
DMA       EQU  80H                ;ADDRESS OF DMA
ENDMA     EQU  01H                ;LAST LOCATION IN DMA
FCB       EQU  005CH              ;ADDRESS OF FCB
CTRLC     EQU  03H                ;CONTROL C MEANS GO TO CPM
GOON      EQU  47H                ;'G' MEANS CONTINUE
DONE      EQU  5AH                ;'Z' MEANS END OF SESSION
QUIT      EQU  14H                ;DC4 SYMBOL MEANS FILE COMPLETE
CONIN     EQU  01H                ;CHECK CONSOLE BUFFER FOR INPUT
CONOUT    EQU  02H                ;OUTPUT CURRENT A REG BYTE TO SCREEN
RIGHTS    DB 'MICROLAN   VERSION 2.1',13,10
          DB 'COPYRIGHT (C) 1985  ROGER D. JASKOT AND HAROLD W. HENRY$'
ENTER     DB 'ENTER NAME OF FILE TO BE SENT. IF THE FILE IS ON',13,10
          DB 'A DISK  IN ANOTHER DRIVE, ENTER IN THE FORMAT:',13,10,10
          DB '             B:FILENAME.FILETYPE$'
```

```
WCHDSK   DB 'WRITE FILE TO WHICH DISK DRIVE?  ENTER AN A FOR A DRIVE,'13,10
         DB 'A B FOR B DRIVE,... OR PRESS RETURN FOR DEFAULT DRIVE.$'
RXMODE   DB 'IN RECEIVE MODE.$'
FNFDMSG  DB 'FILE DOES NOT EXIST, RETURNING TO CPM.$'
TXING1   DB 'TRANSMITTING FILE.$'
HASFILE  DB 'RXING MICRO HAS FILE ALREADY, GOING TO CPM.$'
FULMSG   DB 'RXING MICRO DISK FULL.  RETURNING TO CPM.$'
WELCUM   DB 'WELCOME!',13,10,10
         DB 'YOU ARE NOW ENTERING THE TRANSFER ZONE!$'
INSTRC   DB 'ENTER AN S FOR TRANSMIT MODE, AN R FOR RECEIVE MODE,',13,10
         DB 'OR AN X TO EXIT.$'
FNDMSG   DB 'FILE ALREADY EXISTS. RETURNING TO CPM.$'
EOFMSG   DB 'FILE TRANSMISSION COMPLETED.$'
NOMSG    DB 'NO FILE TRANSFER. RETURNING TO CPM.$'
ABRTMSG  DB 'XMITING MICRO ABORTED FILE TRANSFER.',13,10
         DB 'PLEASE ERASE FILENAME FROM YOUR DIRECTORY.$'
FULLMSG  DB 'DISK FULL. FILE TRANSFER INCOMPLETE.$'
ERMSG    DB 'ENTER FILENAME AGAIN. END WITH <CR>$'
RXING1   DB 'CONNECTION MADE.$'
CONBUF   DB 16                ;BUFFER FOR FILENAME
         DB 00
         RS 16
BAUDMSG  DB 'SELECT BAUD RATE',CDH,0AH
         DB '1 = 300  BAUD',0DH,0AH
```

```
        DB '2 = 600  BAUD',0DH,0AH
        DB '3 =1200  BAUD',0DH,0AH
        DB '4 =2400  BAUD',0DH,0AH
        DB '5 =4800  BAUD',0DH,0AH
        DB '6 =9600  BAUD$'
ERR1    DB 'BAUDRATE OUT OF RANGE$'
BAUD    RW 1
TABL    DB 80H,01,0C0H,00,60H,00,3AH,00,18H,00,0CH,00
```

NOTE: This version of MICROLAN was obtained by direct conversion through the TRANS86 conversion program. We did not attempt to streamline the procedures or efficiency of operation for this IBM compatible version.

# APPENDIX E

## MS.DOS

```
STACK     SEGMENT  PARA  STACK 'STACK'
          DB  512   DUP(0)
STACK     ENDS
DATA      SEGMENT  PARA  PUBLIC   'DATA'
DTA       DB  80H DUP(0)
FCB       DB  36 DUP(0)
ATTN      EQU 52H             ;'R'
RXACK     EQU 72H             ;'r'
DATA1     EQU 03F8H
TXRDY     EQU 20H
RXRDY     EQU 01H
STATUS    EQU 03FDH
TXSYM     EQU 12H             ;DC2 SYMBOL
TXACK     EQU 74H             ;'t'
RLDTA     EQU 0CBH
GOOD      EQU 67H             ;'g'
BAD       EQU 62H             ;'b'
DSKFUL    EQU 64H             ;'d'
DMA       EQU DTA             ;ADDRESS OF DMA
ENDMA     EQU 01H             ;LAST LOCATION IN DMA
CTRLC     EQU 03H             ;CONTROL C MEANS GO TO CPM
```

```
GOON     EQU 47H              ;'G' MEANS CONTINUE
DONE     EQU 5AH              ;'Z' MEANS END OF SESSION
QUIT     EQU 14H              ;DC4 SYMBOL MEANS FILE COMPLETE
CONIN    EQU 01H              ;CHECK CONSOLE BUFFER FOR INPUT
CONOUT   EQU 02H              ;OUTPUT CURRENT A REG BYTE TO SCREEN
RIGHTS   DB 'MICROLAN   VERSION 2.2',13,10
         DB 'COPYRIGHT (C) 1985  ROGER D. JASKOT AND HAROLD W. HENRY$'
ENTER    DB 'ENTER NAME OF FILE TO BE SENT. IF THE FILE IS ON',13,10
         DB 'A DISK IN THE OTHER DRIVE, ENTER IN THE FORMAT:',13,10,10
         DB '             B:FILENAME.FILETYPE$'
WCHDSK   DB 'WRITE FILE TO WHICH DISK DRIVE?  ENTER AN A FOR A DRIVE,',13,10
         DB 'B FOR B DRIVE..., OR PRESS RETURN FOR DEFAULT DRIVE.$'
RXMODE   DB 'IN RECEIVE MODE.$'
FNFDMSG  DB 'FILE DOES NOT EXIST, RETURNING TO CPM.$'
TXING1   DB 'TRANSMITTING FILE.$'
HASFILE  DB 'RXING MICRO HAS FILE ALREADY, GOING TO CPM.$'
FULMSG   DB 'RXING MICRO DISK FULL.  RETURNING TO CPM.$'
WELCUM   DB 'WELCOME!',13,10,10
         DB 'YOU ARE NOW ENTERING THE TRANSFER ZONE$'
INSTRC   DB 'ENTER AN S FOR TRANSMIT MODE, AN R FOR RECEIVE MODE,',13,10
         DB 'OR AN X TO EXIT.$'
FNDMSG   DB 'FILE ALREADY EXISTS. RETURNING TO CPM.$'
EOFMSG   DB 'FILE TRANSMISSION COMPLETED.$'
NOMSG    DB 'NO FILE TRANSFER. RETURNING TO CPM.$'
```

```
ABRTMSG  DB  'XMITING MICRO ABORTED FILE TRANSFER.',13,10
         DB  'PLEASE ERASE FILENAME FROM YOUR DIRECTORY.$'
FULLMSG  DB  'DISK FULL. FILE TRANSFER INCOMPLETE.$'
ERMSG    DB  'ENTER FILENAME AGAIN. END WITH <CR>$'
RXING1   DB  'CONNECTION MADE.$'
CONBUF   DB  16H         ;BUFFER FOR FILENAME
         DB  00
         DB  16H   DUP(0)
BAUDMSG  DB  'SELECT BAUD RATE',0DH,0AH
         DB  '1 = 300  BAUD',0DH,0AH
         DB  '2 = 600  BAUD',0DH,0AH
         DB  '3 =1200  BAUD',0DH,0AH
         DB  '4 =2400  BAUD',0DH,0AH
         DB  '5 =4800  BAUD',0DH,0AH
         DB  '6 =9600  BAUD$'
ERR1     DB  'BAUDRATE OUT OF RANGE$'
BAUD     DW  00H
TABL     DB  80H,01,0C0H,00,60H,00,3AH,00,18H,00,0CH,00
DATA  ENDS
CODE  SEGMENT  PARA  PUBLIC   'CODE'
START PROC  FAR
     ASSUME CS:CODE
     PUSH DS
     MOV AX,0
```

```
      PUSH AX
      MOV AX,DATA
      MOV ES,AX
      ASSUME ES:DATA
      MOV SI,80H
      MCV DI,OFFSET DTA
      MOV CX,80H
      REP MOVSB
      MOV SI,005CH
      MCV DI,OFFSET FCB
      MOV CX,12
      REP MOVSB                ;TRANSFER BOTH PARAMETER AREAS TO OUR SGMENT
      MOV DS,AX
      ASSUME DS:DATA
      MOV DX,OFFSET BAUDMSG    ;BAUDRATE HEADER
      MCV AH,09H               ;PRINT SAME
      CALL BDOS
      MOV AH,01H               ;GET KEYBOARD INPUT
      CALL BDOS
      SUB AL,31H               ;CONVERT TO TABLE OFFSET
      CMP AL,05H
      JBE SETB1
      JMP ERROR1
SETB1: MCV BX,OFFSET TABL
```

```
    ADD AL,AL
    MOV AH,0
    ADD BX,AX
    MOV DX,[BX]
    MOV BX,OFFSET BAUD
    MOV [BX],DX
    MOV DX,03FBH              ;LINE CONTROL
    MOV AL,83H                ;DLAB=1
    OUT DX,AL
    MOV DX,03F8H              ;BAUDRATE DIVISOR
    MOV BX,OFFSET BAUD
    MOV AX,[BX]
    OUT DX,AX
    MOV DX,03FBH              ;CONTROL
    MOV AL,03H
    OUT DX,AL                 ;RESET DLAB
    MOV AH,1AH                ;OPEN THE DTA
    MOV DX,OFFSET DTA
    CALL BDOS
INIT: CALL CRLF
    MOV   AH,09H              ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET RIGHTS    ;COPYRIGHTS AND NAMES OF AUTHORS
    CALL BDOS
    CALL CRLF
```

```
     CALL CRLF
     MOV   AH,09H             ;PRINT STRING TO SCREEN
     MOV   DX,OFFSET WELCUM   ;WELCOME MSG
     CALL BDOS
     CALL CRLF
     CALL CRLF
     MOV   AH,09H             ;PRINT STRING TO SCREEN
     MOV   DX,OFFSET INSTRC   ;SEND, RECEIVE, OR QUIT?
     CALL BDOS
     CALL CRLF
     CALL CRLF
HOLDING: MOV   AH,06H         ;CHECK FOR CONSOLE INPUT
     MOV   DI,0FFH            ;LOOKING FOR INPUT
     CALL BDOS
     AND  AL,0DFH             ;ENSURE LETTER IS A CAPITAL
     CMP  AL,53H              ;IS IT AN 'S'?
     JNE G1
     JMP MASTER               ;IF SO, START FILE TRANSFER
 G1: CMP  AL,52H              ;IS IT AN 'R'?
     JNE G2
     JMP SLAVE                ;IF SO, PREPARE TO RECEIVE FILE
 G2: CMP  AL,53H              ;IS IT AN 'X'?
     JNE G3
     JMP CPM                  ;IF SO, GO TO CPM
```

```
G3: JMP  HOLDING                ;REPEAT UNTIL INPUT FOUND
MASTER: MOV   AH,09H            ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET ENTER       ;ENTER FILENAME
    CALL BDOS
    CALL CRLF
FILLUP: MOV   BX,OFFSET FCB     ;ADDRESS OF FCB
    MOV  [BX],BYTE PTR 00H
    INC  BX
    MOV   CL,0BH                ;11 SPACES
FILLUP1: MOV  [BX],BYTE PTR 20H ;FILL MEMORY ADDRESS WITH SPACES
    INC  BX                     ;MOVE PTR TO NEXT ADDRESS
    DEC  CL                     ;DECREMENT COUNTER
    JNZ  FILLUP1                ;REPEAT UNTIL DONE
    MOV   CL,13H                ;TOTAL OF 20 SPACES
FILLUP2: MOV [BX],BYTE PTR 00H ;FILL REST IF ADDRESS WITH 0's
    INC  BX                     ;MOVE PTR TO NEXT ADDRESS
    DEC  CL                     ;DECREMENT COUNTER
    JNZ  FILLUP2                ;REPEAT UNTIL DONE
HOLD1: MOV   AH,0AH             ;READ CONSOLE BUFFER
    MOV   DX,OFFSET CONBUF      ;ADDRESS OF FIRST LETTER OF FILENAME
    CALL BDOS
    MOV   BX,OFFSET CONBUF      ;ADDRESS OF CONSOLE BUFFER
    MOV   DX,OFFSET FCB+1       ;FCB ADDRESS
    INC  BX
```

```
    MOV  CH,[BX]            ;STORE COUNT IN BX REGISTER
    MCV  AL,[BX]            ;MCVE COUNT TO ACCUMULATOR
    OR   AL,AL              ;IS THERE AN INPUT?
    JNE G4
    JMP ERROR               ;TRY AGAIN
G4: INC  BX
FLUP: MOV  AL,[BX]
    CMP  AL,3AH             ;IS CHARACTER A ':'?
    JNE G5
    JMP DSKSEL              ;IF SO, GO TO DISK SELECT
G5: CMP  AL,2EH             ;IS IT A '.'?
    JNE G6
    JMP FIXIT               ;IF SO, SKIP TO FILETYPE
G6: CMP  AL,40H             ;CHECK FOR LETTER
    JNC G7
    JMP DONTFIX             ;SKIP NEXT STEP IF NOT LETTER
G7: AND  AL,0DFH            ;ENSURE LETTER IS A CAPITAL
DONTFIX: XCHG BX,DX
    MOV [BX],AL
    XCHG BX,DX              ;STORE LETTER IN FCB
    INC  DX
    INC  BX
    DEC  CH
    JNZ  FLUP               ;REPEAT UNTIL END
```

```
LISN1: MOV   AL,ATTN            ;LETTER 'R'
    CALL POUT1
    MOV   CL,03H                ;LISTEN 3 TIMES
LISN: CALL PIN
    CMP  AL,PXACK               ;IS IT AN 'r'?
    JNE G8
    JMP XMIT                    ;IF SO, THEN XMIT
G8: DEC  CL                     ;OTHERWISE DECREMENT CTR
    JNZ  LISN                   ;LISTEN UNTIL CTR IS ZERO
    JMP  LISN1                  ;THEN TRY AGAIN
XMIT: CALL CRLF
    CALL CRLF
    MOV   AH,09H                ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET RXING1      ;'r' WAS RECEIVED
    CALL BDOS
    CALL CRLF
XMIT1: MOV   AL,TXSYM           ;DC2 SYMBOL FOR SYNC AT START
    CALL POUT1                  ;OF 128 BYTE BLOCK
    MOV   CL,08AH               ;LISTEN 138 TIMES
LITTLET: CALL PIN
    CMP  AL,TXACK               ;WAS 't' RECEIVED?
    JNE G9
    JMP TXFCB                   ;IF SO, XMIT FILE CTRL BLK
G9: DEC  CL                     ;OTHERWISE KEEP LISTENING
```

```
    JNZ  LITTLET                ;UNTIL CTR IS ZERO,
    JMP  XMIT1                  ;THEN SEND DC2 SYNC AGAIN
TXFCB: CALL CRLF                ;SEND FILENAME TO RXING MICRO
    CALL OPENIT                 ;SEE IF FILE EXISTS.  IF SO, OPEN IT
TXFCB1: MOV   CH,00H            ;INITIALIZE CHECKSUM REGISTER
    MOV   BX,OFFSET FCB         ;SET PTR TO 1ST LETTER IN FILENAME
FCBLUP: MOV  AL,CH              ;PERFORM CHECKSUM OPERATION
    INC  BX                     ;MOVE PTR TO NEXT BYTE
    XOR  AL,[BX]                ;BY XORING CURRENT BYTE
    MOV  CH,AL                  ;WITH B REGISTER
    MOV  AL,[BX]                ;PUT CURRENT BYTE IN ACCUM
    CALL POUT1                  ;SEND CURRENT BYTE
    CMP  AL,0H                  ;CHECK FOR END OF FILENAME
    JNE G10
    JMP FCBCK                   ;IF END, GO TO CHECKSUM LOOP
G10: JMP  FCBLUP                ;IF NOT, REPEAT FCB LOOP
FCBCK: MOV   CL,20H             ;LOOP 32 TIMES
FCBCK1: CALL PIN                ;FOR SYNC WITH SLAVE
    CMP  AL,RXACK               ;IS IT AN 'r'?
    JNZ  FCBCK1                 ;IF NOT, LISTEN AGAIN.  IF SO,
    MOV  AL,CH                  ;PUT CHECKSUM IN ACCUM
    CALL POUT                   ;SEND CHECKSUM
    PUSH CX                     ;SAVE CHECKSUM
    MOV   AL,0H                 ;CLEAR ACCUM
```

```
    MOV   CH,80H              ;LISTEN 100 TIMES
FCBTMCT: CALL PIN             ;READ MAIL
    CMP  AL,BAD               ;DID IT CHECK BAD?
    JNE G11
    JMP RSNDFCB               ;IF SO, SEND FCB AGAIN
G11: CMP  AL,GOOD             ;DID IT CHECK GOOD?
    JNE G12
    JMP WAITFIL               ;IF SO, GO TO NEXT ROUTINE
G12: DEC  CH                  ;IF NOT, DECREMENT CTR, AND
    JNZ  FCBTMOT              ;IF NOT 0, LISTEN AGAIN
    POP  CX                   ;CLEAR STACK
    DEC  CL                   ;IF SO, DECREMENT CL
    JNZ  FCBCK1               ;AND REPEAT UNTIL CL=0
    JMP  TXFCB                ;IF 0, ASSUME PROBLEM AND SEND AGAIN
WAITFIL: POP  CX              ;CLEAR STACK
WAIT2: MOV   CX,07FFH         ;COUNT LOOP APPX 2K
WAIT1: CALL STATIN1           ;ANY 'MAIL'?
    JZ   WAIT1                ;IF NOT, CHECK AGAIN
    DEC  CX                   ;IF SO, DECREMENT CTR
    JNE G13
    JMP GOCPM                 ;AND, IF 0, QUIT
G13: CALL PIN                 ;OTHERWISE READ 'MAIL'
    CMP  AL,QUIT              ;DOES RXING MICRO ALREADY HAVE FILE?
    JNE G14
```

```
    JMP GOCPM1                  ;IF SO, GO TO CPM
G14: CMP  AL,GOON               ;IS IT THE GO ON SIGNAL 'G'
    JNZ  WAIT2                  ;IF NOT, LISTEN AGAIN.  ALLOW RXING
                                ;MICRO TO CATCH UP
    CALL CRLF
TXDATA: MOV   AH,09H            ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET TXING1      ;SAYS FILE BEING SENT
    CALL BDOS
    CALL CRLF
RDSEQ: CALL READSEQ             ;READ FIRST 128 BYTE BLOCK
SEND: CALL CHECK                ;AND SEND TO RXING MICRO
    MOV   AL,TXSYM              ;DC2 SYMBOL FOR SYNC AT START OF DATA
    CALL POUT1
    MOV   CL,0FH                ;LISTEN 15 TIMES
LITLET2: CALL PIN
    CMP  AL,TXACK               ;IS IT A 't'?
    JNE G15
    JMP SLUP2                   ;IF SO, READY TO SEND DATA
G15: CMP  AL,DSKFUL             ;IS RXING MICRO'S DISK FULL?
    JNE G16
    JMP FULDISK                 ;IF SO, QUIT
G16: DEC  CL                    ;IF NOT, DECREMENT CTR
    JNZ  LITLET2                ;LISTEN AGAIN, UNLESS CTR IS 0,
    JMP  SEND                   ;THEN TRY TO SYNC AGAIN
```

```
SLUP2: MOV AL,RLDTA              ;0CBH MEANS TIME FOR DATA
    CALL POUT1
    MOV CX,07FFH                 ;WAIT LOOP APPX 2K
SLUP3: CALL PIN
    CMP AL, RLDTA                ;IS IT ECHO?
    JZ SLUP1                     ;IF SO, SEND DATA
    DEC CX                       ;DECREMENT COUNTER
    JNZ SLUP3                    ;REPEAT UNTIL ZERO
    JMP SLUP2
SLUP1: MOV   BX,OFFSET DTA       ;POINTER TO 1ST INFO BYTE
    MOV   CH,00H                 ;INITIALIZE CHECKSUM LOCATION
    MOV CL,80H
SLOOP: MOV  AL,CH                ;PERFORM CHECKSUM
    XOR  AL,[BX]                 ;XOR DATA WITH CH REGISTER
    MOV  CH,AL
    MOV  AL,[BX]                 ;PUT BYTE IN ACCUMULATOR
    CALL POUT                    ;DATA IS TRANSFERRED
    INC  BX                      ;MOVE PTR TO NEXT BYTE
    DEC CL
    JNZ SLOOP
CRC: MOV  AL,CH                  ;PUT CHECKSUM IN ACCUMULATOR
    CALL POUT                    ;AND SEND TO RXING MICRO
CRCTMCT: MOV   CH,01AH           ;LISTEN 26 TIMES
CRCT1: CALL STATIN1              ;CHECK INPUT BUFFER
```

```
    JZ   CRCT1                 ;IF NOTHING, TRY AGAIN
    CALL PIN                   ;READ MAIL
    CMP  AL,BAD                ;IS CHECK BAD?
    JNE G17
    JMP RESEND                 ;IF SO, SEND BLOCK AGAIN
G17: CMP  AL,GOOD              ;IS CHECK GOOD?
    JNE G18
    JMP RDSQRPT                ;IF SO, READ NEXT BLOCK
G18: DEC  CH                   ;DECREMENT COUNTER
    JNZ  CRCT1                 ;IF NOT TIMED OUT,LISTEN AGAIN
    JMP  SEND                  ;IF TIMED OUT, ASSUME PROBLEM.
                               ;SEND BLOCK AGAIN
DSKSEL: MOV   BX,OFFSET CONBUF+2  ;ADDRESS OF DISK SEL ENTRY
    MOV  AL,[BX]               ;PUT DISK SEL IN ACCUM
    AND  AL,ODFH               ;ENSURE LETTER IS CAPITAL
    CMP  AL,'A'                ;IS LETTER AN 'A'?
    JZ ADISK                   ;IF SO, SET FOR A DRIVE.
    CMP  AL,'B'                ;IS LETTER A 'B'?
    JZ BDISK                   ;IF SO, SET FOR B DRIVE.
    CMP AL,'C'                 ;IS LETTER A 'C'?
    JZ CDISK                   ;IF SO, SET FOR C DRIVE.
    JMP DSKSEL1                ;IF NEITHER, RETURN TO FILENAME LOOP.
ADISK: MOV   DI,OFFSET FCB     ;SET PTR TO DRIVE BYTE.
    MOV  [DI],BYTE PTR 01H     ;SET FCB FOR A DRIVE.
```

```
    JMP  DSKSEL1                ;RETURN TO FILENAME LOOP.
BDISK: MOV   DI,OFFSET FCB      ;SET PTR TO DRIVE BYTE.
    MOV  [DI],BYTE PTR 02H      ;SET FCB FOR B DRIVE.
    JMP  DSKSEL1                ;RETURN TO FILENAME LOOP.
CDISK: MOV   DI,OFFSET FCB      ;SET PTR TO DRIVE BYTE.
    MOV  [DI],BYTE PTR 03H      ;SET FCB FOR C DRIVE.
DSKSEL1: INC  BX                ;MOVE BUFFER POINTER TO FILENAME.
    INC BX
    MOV   DX,OFFSET FCB+1       ;FCB FILENAME ADDRESS.
    JMP  FLUP                   ;RETURN TO FILENAME LOOP.
RESEND: MOV   AH,CONCUT         ;PRINT TO SCREEN
    MOV   DL,BAD                ;A 'b' IF CHECKSUM WAS BAD
    CALL BDOS
    JMP  SEND                   ;AND SEND BLOCK AGAIN
RSNDFCB: POP  CX                ;RECALL CHECKSUM
    MOV  AL,CH                  ;PUT CHECKSUM IN ACCUM
    ADD  AL,3                   ;ADD 3 TO OFFSET
    CALL POUT1                  ;SEND BYTE
RSNDF: CALL PIN
    CMP  AL,RXACK               ;IS IT AN 'r'
    JNZ  RSNDF                  ;IF NOT LISTEN AGAIN
RSNDFC1: CALL PIN               ;READ MAILBOX
    CMP  AL,TXACK               ;SYNC WITH RXING MICRO
    JNZ  RSNDFC1                ;REPEAT UNTIL TXACK RECEIVED
```

```
    JMP  TXFCB1                 ;IF SO, RESEND FCB
FIXIT: MOV   DX,OFFSET FCB+9 ;MOVE POINTER TO FILETYPE AREA
    INC  BX                     ;MOVE PTR TO FIRST LETTER OF FILTYPE
    JMP  FLUP
ERROR: MOV   AH,09H             ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET ERMSG       ;ERROR MESSAGE
    CALL BDOS
    CALL CRLF
    JMP  HOLD1                  ;LOOK FOR INPUT AGAIN
POUT1 PROC NEAR
    LAHF
    PUSH AX                     ;SEND THE DATA
                                ;FIRST, SAVE THE CURRENT BYTE
    MOV   AH,06H                ;CHECK FOR CONSOLE INPUT
    MOV   DL,OFFH               ;LOOKING FOR INPUT
    CALL BDOS
    CMP  AL,CTRLC               ;IS THERE A CONTROL C?
    JNE G22
    JMP STOPS
G22: CALL CHECK                 ;PERFORM CHECK
    POP AX
    SAHF                        ;AND RECALL BYTE
    MOV DX,DATA1
    OUT DX,AL
```

```
    RET
POUT1 ENDP
OPENIT   PROC NEAR
    MOV   AH,0FH              ;OPEN FILE CODE
    MOV   DX,OFFSET FCB       ;FILE CTRL BLOCK ADDRESS IN DX REG PR
    CALL BDOS
    CMP  AL,0FFH              ;FF = FILE NOT FOUND
    JNE G23
    JMP FNFOUND               ;IF FILE NOT FOUND
G23: RET                      ;OTHERWISE, RET TO TX DATA
OPENIT   ENDP
CLOSIT: MOV   AH,10H          ;CLOSE FILE CODE
    MOV   DX,OFFSET FCB       ;FILE CTRL BLOCK ADDRESS IN DE REGPR
    CALL BDOS
CLOSIT1: MOV   AL,DONE        ;END OF SESSION MSG 'Z'
    CALL POUT1                ;SEND TO RXING MICRO
    MOV   AL,0H               ;CLEAR ACCUM
    CALL PIN                  ;CHECK REPLY
    CMP  AL,DONE              ;DOES RXING MICRO AGREE?
    JNZ  CLOSIT1              ;IF NOT, REPEAT
    JMP  GOCPM                ;IF SO, GO TO CPM
READSEQ  PROC NEAR
    PUSH CX
    PUSH DX
```

```
    MOV   AH,14H              ;READ SEQUENTIAL CODE
    MOV   DX,OFFSET FCB       ;FILE CTRL BLOCK ADDRESS IN DX REGPR
    CALL BDOS
    POP DX
    POP CX
    CMP  AL,0                 ;0 MEANS SUCCESSFUL READ
    JZ G24
    JMP EOFILE                ;IF NOT 0, ASSUME FINISHED WITH FILE
G24: RET
    READSEQ  ENDP
RDSQRPT: MOV   AH,CONOUT      ;PRINT TO SCREEN
    MOV   DL,02AH             ;'*' SO USER KNOWS BLK WAS SENT
    CALL BDOS
    JMP  RDSEQ                ;TO READ NEXT 128 BYTE BLK
FNFOUND: MOV   AL,QUIT        ;TELL RXING MICRO NO FILE FOUND
    CALL POUT1
    MOV   AH,09H              ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET FNFDMSG   ;FILE NOT FOUND MSG
    CALL BDOS
    CALL CRLF
    JMP  GOCPM                ;AND GO TO CPM
EOFILE: POP AX                ;CORRECT STACK POINTER
EOFILE2: MOV   AL,TXSYM       ;DC2 SYMBOL FOR SYNC WITH RXING MICRO
    CALL POUT1
```

```
    MOV   CL,0FH              ;LISTEN 15 TIMES
LITLET3: CALL STATIN1         ;CHECK FOR MAIL
    JZ   LITLET3              ;IF NONE, CHECK AGAIN
    CALL PIN                  ;READ MAIL
    CMP  AL,TXACK             ;IS IT A 't'?
    JNE G25
    JMP EOFIL1                ;IF SO, CONTINUE
G25: DEC  CL                  ;IF NOT, DECREMENT COUNTER
    JNZ  LITLET3              ;AND LISTEN AGAIN, UNLESS COUNTER IS
    JMP  EOFILE2              ;0.  THEN TRY AGAIN
EOFIL1: MOV   AL,QUIT         ;DC4 SYMBOL.  TELLS RXING MICRO THAT
    CALL POUT1                ;THE FILE IS DONE
    CALL PIN                  ;LISTEN FOR REPLY
    CMP  AL,QUIT              ;DOES RXING MICRO ACKNOWLEDGE?
    JNZ  EOFIL1               ;IF NOT, TRY AGAIN
    CALL CRLF
    MOV   AH,09H              ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET EOFMSG    ;IF SO, TELL USER FILE IS DONE
    CALL BDOS
    CALL CRLF
    JMP  CLOSIT               ;AND CLOSE THE FILE
STOPS: MOV   AL,CTRLC         ;SEND CTRLC TO RXING MICRO
    MOV DX,DATA1
    OUT  DX,AL
```

```
    MOV AL,0H                   ;CLEAR ACCUM
    CALL PIN                    ;FROM RXING MICRO
    CMP  AL,CTRLC               ;ACK FROM RXING MICRO
    JNZ  STOPS                  ;REPEAT UNTIL ACK
    POP AX
    JMP  GOCPM
FULDISK: MOV   AL,DONE          ;LETTER 'Z' TO ACKNOWLEDGE
    CALL POUT1                  ;SEND BYTE
    MOV   AH,09H                ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET FULMSG      ;SAYS RXER'S DISK FULL
    CALL BDOS
    CALL CRLF
    JMP  GOCPM
GOCPM: MOV   AL,0H              ;RESET THE ACCUMULATOR AND
    MOV DX,DATA1
    OUT  DX,AL                  ;CLEAR OUTPUT BUFFER
    CALL CRLF
    CALL PIN
    JMP  CPM                    ;AND GO TO CPM
GOCPM1: MOV   AH,09H            ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET HASFILE     ;RXING MICRO HAS FILE ALREADY
    CALL BDOS
    JMP  GOCPM
SLAVE: MOV   AH,09H             ;PRINT STRING TO SCREEN
```

```
    MOV   DX,OFFSET WCHDSK    ;SELECT DISK DRIVE
    CALL BDOS
    CALL CRLF
    CALL CRLF
DRVSEL: MOV   AH,06H          ;CHECK FOR CONSOLE INPUT
    MOV   DL,0FFH             ;LOOKING FOR INPUT
    CALL BDOS
    CMP  AL,0DH               ;IS IT A <CR>?
    JNE G26
    JMP CONT                  ;IF SO, ENTER RECEIVE MODE
G26: AND  AL,0DFH             ;ENSURE LETTER IS A CAPITAL
    CMP  AL,'A'               ;IS IT AN 'A'?
    JZ G27
    JMP DISKB                 ;SKIP TO B IF NOT 'A'
G27: MOV   BX,OFFSET FCB      ;ADDRESS OF DISK DRIVE BYTE
    MOV  [BX],BYTE PTR 01H    ;SET BYTE TO A DISK DRIVE
    JMP  CONT                 ;THEN CONTINUE
DISKB: CMP  AL,'B'            ;IS IT A 'B'?
    JNZ  DISKC                ;SKIP TO C IF NOT 'B'
    MOV   BX,OFFSET FCB       ;ADDRESS OF DISK DRIVE BYTE
    MOV  [BX],BYTE PTR 02H    ;SET BYTE TO B DISK DRIVE
    JMP CONT                  ;THEN CONTINUE
DISKC: CMP AL,'C'             ;IS IT A 'C'?
    JNZ DRVSEL                ;IF NOT, LISTEN AGAIN
```

```
    MOV   BX,OFFSET FCB       ;ADDRESS OF DISK DRIVE BYTE
    MOV  [BX],BYTE PTR 03H    ;SET BYTE TO C DISK DRIVE
CONT: MOV   AH,09H            ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET RXMODE    ;IN RECEIVE MODE
    CALL BDOS
    CALL CRLF
SLAVE1: MOV   AL,00H          ;RESET ACCUMULATOR
    CALL PIN                  ;LISTENING FOR AN 'R'
    CMP  AL,ATTN              ;'R'
    JNZ  SLAVE1               ;IF 'R' RX'D, CONTINUE. IF NOT
                              ;LISTEN AGAIN
    CALL CRLF
    MOV   AH,09H              ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET RXING1    ;CONNECTION MADE
    CALL BDOS
    CALL CRLF
    MOV   AL,RXACK            ;'r'
    CALL POUT                 ;SEND AN 'r' TO XMITING MICRO
LISTEN: CALL PIN1             ;LISTENING FOR A 'DC2'
    CMP  AL,TXSYM             ;'DC2'
    JNZ  LISTEN               ;IF 'DC2' RX'D, CONTINUE. IF NOT,
                              ;LISTEN AGAIN
    CALL CRLF
RXFCB: MOV   BX,OFFSET FCB+1 ;ADDRESS OF FCB MEM LOC INTO BX REG PR
```

```
    MOV   CL,1EH                ;COUNTER FOR FCB'S 31 SPACES
RSTFCB: MOV  [BX],BYTE PTR 00H ;FILL FCB WITH )'S
    INC  BX                     ;MOVE PTR TO NEXT MEMORY ADDRESS IN FCB
    DEC  CL                     ;DECREMENT COUNTER
    MOV  AL,CL
    CMP  AL,0
    JNZ  RSTFCB                 ;IF COUNTER = 0, CONT. IF NOT,
                                ;PUT ANOTHER 0 IN FCB
    MOV   CH,00H                ;INITIALIZE CHECKSUM
    MOV   BX,OFFSET FCB+1       ;LOAD 2ND ADDRESS OF FCB IN
    MOV   AL,TXACK              ;'t'
    CALL POUT                   ;SEND 't' TO XMITING MICRO FOR SYNC
    CALL PIN1                   ;CLEAR THE ACCUMULATOR
RST1: CALL STATIN1              ;CHECKING FOR INPUT
    JZ   RST1
RST2: CALL PIN1                 ;FILE NAME DATA
    CMP  AL,QUIT                ;IS DATA A 'QUIT'?
    JNE G28
    JMP NOFILF                  ;FILE DID NOT EXIST
G28: CMP  AL,0H                 ;CHECK IF FILENAME COMPLETELY SENT
    JNE G29
    JMP FCBCRC                  ;IF FILENAME RX'D, GO TO CHECKSUM
G29: CMP  AL,TXSYM              ;CHECK IF DATA IS VALID
    JZ   RST2                   ;IF DATA IS NOT FILENAME,
```

```
    MOV  [BX],AL              ;PUT FILENAME IN FCB
    CALL OUTPUT               ;PRINT FILENAME TO SCREEN
    MOV  AL,CH
    XOR  AL,[BX]              ;CALCULATE CHECKSUM
    MOV  CH,AL
    INC  BX                   ;MOVE PTR TO NEXT FCB ADDRESS
    JMP  RST1
FCBCRC: MOV   AL,RXACK        ;'r'
    CALL POUT                 ;SYNC DATA WITH XMITING MICRO
FCBCRC1: CALL STATIN1         ;CHECKING FOR INPUT
    JZ   FCBCRC1
    CALL CRLF
    CALL PIN                  ;CHECKSUM DATA
    CMP  AL,CH                ;COMPARE CHECKSUM
    JNE G30
    JMP STRTFIL               ;CHECKSUM MATCHED
G30: ADD  AL,3                ;ADD 3 TO THE CHECKSUM
    MOV  CL,AL                ;STORE IN REGISTER
    MOV   AL,BAD              ;CHECKSUM DID NOT MATCH
    CALL POUT                 ;TELL XMITING MICRO
CLEAR: CALL PIN
    CMP  AL,CL                ;XMITING MICRO STOPPED SENDING CHKSUM?
    JNZ  CLEAR                ;IF NOT,LISTEN AGAIN
    MOV   AL,RXACK            ;SYNC WITH XMITING MICRO
```

```
    CALL POUT
    JMP  RXFCB                  ;TRY AGAIN
STRTFIL: MOV   AL,GOOD          ;READY TO CHECK IF FILE ALREADY PRESENT
    CALL POUT
    CALL OPNFILE                ;CHECK IF FILE EXISTS
    CALL MAKEFIL                ;CREATE NEW FILE
RXD1: MOV   CH,00H              ;INITIALIZE CHECKSUM
    MOV   BX,OFFSET DTA         ;LOAD ADDRESS OF DMA MEM LOC TO
                                ;BX REGISTER PAIR
    MOV   CL,81H                ;INITIALIZE COUNTER WITH SIZE OF DMA
RXDS: CALL STATIN1
    JZ RXDS
RXD2: CALL PIN1                 ;SYNC WITH XMITING MICRO
    CMP  AL,TXSYM               ;COMPARE WITH 'DC2'
    JNZ  RXD2
    MOV   AL,TXACK              ;'T'
    CALL POUT                   ;IN SYNC WITH XMITING MICRO
RXDS1: CALL STATIN1             ;CHECKING FOR INPUT
    JZ   RXDS1
RXYET: CALL PIN1
    CMP AL,RLDTA                ;IS IT OCBH?
    JZ RXYET1                   ;IF SO, GO TO RECEIVE DATA
    CMP AL,QUIT                 ;IS IT 'DC4' FOR QUIT?
    JNZ RXYET4                  ;IF NOT, LISTEN AGAIN; OTHERWISE,
```

```
    JMP CLSFILE                 ;IF SO, CLOSE FILE
RXYET4: JMP RXDS1               ;LISTEN AGAIN
RXYET1: MOV AL,RLDTA            ;ACK REAL DATA COMING
    CALL POUT
    MOV AL,00H                  ;CLEAR ACCUM
RXYET2: CALL STATIN1            ;CHECKING FOR INPUT
    JZ RXYET2
RXYET3: CALL PIN                ;READ DATA
    CMP AL,RLDTA                ;IS IT STILL RLDTA?
    JZ RXYET3
RXD3: DEC  CL                   ;DECREMENT COUNTER
    JNE G32
    JMP RXCRC                   ;CHECKSUM RX'D
G32: MOV [BX],AL                ;PUT THE DATA IN MEMORY
    MOV  AL,CH
    XOR  AL,[BX]                ;CALCULATE CHECKSUM
    MOV  CH,AL
    INC  BX                     ;MOVE PTR TO NEXT DMA ADDRESS
RXD4: CALL STATIN1              ;CHECK FOR INPUT
    JZ RXD4                     ;LOOP UNTIL INPUT
    CALL PIN
    JMP  RXD3
RXCRC: MOV  AL,AL               ;ENSURE CH IS COMPARED TO A
    CMP  AL,CH                  ;COMPARE WITH CHECKSUM
```

```
    JNE G33
    JMP WRITFIL                 ;128 BYTE BLOCK SENT
G33: MOV   AL,BAD               ;CHECKSUM DID NOT MATCH
    CALL POUT                   ;NOTIFY XMITING MICRO
    JMP  RXD2                   ;SEND 128 BYTE BLOCK AGAIN
POUT  PROC NEAR
    LAHF
    PUSH AX                     ;SAVE THE DATA
    CALL CHECK
    POP AX
    SAHF                        ;RETURN THE DATA
    MOV DX,DATA1
    OUT DX,AL                   ;SEND DATA
    RET
POUT  ENDP
PIN1  PROC NEAR
    MOV DX,DATA1
    IN AL,DX
    CMP  AL,CTRLC               ;DID XMITING MICRO ABORT?
    JNE G34
    JMP ABORT                   ;IF SO, ABORT
G34: RET
PIN1  ENDP
WRITFIL: MOV   AL,GOOD          ;XMIT THAT THE CHECKSUM IS CORRECT
```

```
    CALL POUT
    CALL WRITSEQ              ;START WRITING FILE TO DISK
    MOV   AH,CONOUT           ;PRINT TO SCREEN
    MCV   DL,02AH             ;'*' TO PRINT TO SCREEN
    CALL BDOS
    JMP  RXD1
OPNFILE  PROC NEAR
    MCV   AH,0FH              ;OPEN FILE CODE
    MOV   DX,OFFSET FCB       ;FCB ADDRESS IN DX RGSTR PAIR
    CALL BDOS
    CMP  AL,0FFH              ;FF = FILE NOT FOUND
    JZ G35
    JMP FILFND                ;FILE EXISTS
G35: RET
OPNFILE  ENDP
CLSFILE: MOV   AL,QUIT        ;'EC4'
    CALL POUT                 ;AGREE END OF FILE
    MOV   AH,10H              ;CLOSE FILE CODE
    MOV   DX,OFFSET FCB       ;FCB ADDRESS IN DX RGSTR PAIR
    CALL BDOS
    CALL CRLF
    MOV   DX,OFFSET ECFMSG    ;FILE TRANSMISSION COMPLETED
    MOV   AH,09H              ;PRINT STRING TO SCREEN
    CALL BDOS
```

```
    CALL CRLF
    MOV   AL,0H               ;CLEAR THE ACCUMULATOR
CLSFIL1: CALL PIN1            ;LOOKING FOR END OF SESSION MSG
    CMP  AL,DONE              ;'Z' = END OF SESSION
    JNZ  CLSFIL1
    MOV   AL,DONE             ;END OF SESSION MESSAGE
    CALL POUT                 ;CONFIRM RECEPTION OF E-O-SESSION MSG
    JMP  CPM
MAKEFIL  PROC NEAR
    MOV   AH,16H              ;MAKE NEW FILE CODE
    MOV   DX,OFFSET FCB       ;FCB ADDRESS IN DX RGSTR PAIR
    CALL BDOS
    MOV   AL,GOON             ;CONTINUE MESSAGE
    CALL POUT
    RET                       ;RETURN TO RX FIRST 128 BYTE BLOCK
MAKEFIL  ENDP
WRITSEQ  PROC NEAR
    PUSH CX
    PUSH DX
    MOV   AH,15H              ;WRITE THE FILE TO THE DISK
    MOV   DX,OFFSET FCB       ;FCB IN DX RGSTR PAIR
    CALL BDOS
    POP DX
    POP CX
```

```
    OR   AL,AL                  ;CHECK IF DISK IS FULL
    JZ G36
    JMP FULLDSK                 ;IF SO, JUMP TO FULLDSK
G36: RET
WRITSEQ  ENDP
FILFND: MOV   AL,QUIT           ;TELL XMITING MICRO, FILE FOUND
    CALL POUT
    MOV   AH,09H                ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET FNDMSG      ;FILE ALREADY EXISTS. GO TO CPM
    CALL BDOS
    CALL CRLF
    JMP  CPM
NOFILE: MOV   AH,09H            ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET NOMSG       ;NO FILE TRANSFER
    CALL BDOS
    CALL CRLF
    JMP  CPM
ABORT: CALL CRLF
    MOV   AL,CTRLC              ;SEND XMITING MICRO ABORT ACK
    CALL POUT
    MOV   AH,09H                ;PRINT STRING TO SCREEN
    MOV   DX,OFFSET ABRTMSG     ;XMITING MICRO ABORTED
    CALL BDOS
    CALL CRLF
```

```
    JMP  GOCPM                      ;GO TO CPM
  FULLDSK: MOV   AL,DSKFUL          ;'I'
    CALL POUT                       ;TELL XMITING MICRO DISK FULL
    CALL PIN                        ;AWAITING CONFIRMATION
    CMP  AL,DONE
    JNZ  FULLDSK
    MOV   AH,09H                    ;PRINT TO SCREEN
    MOV   DX,OFFSET FULLMSG         ;FILE TRANSFER INCOMPLETE, DISK FULL
    CALL BDOS
    CALL CRLF
    JMP  CPM
CHECK PROC NEAR
    CALL STATIN2                    ;CHECK STATUS BYTE
    JZ   CHECK                      ;CONTINUE UNTIL TXRDY IS SET
    RET
CHECK ENDP
STATIN1  PROC NEAR
    MOV DX,STATUS
    IN AL,DX
    AND AL,RXRDY
    RET
STATIN1  ENDP
STATIN2  PROC NEAR
    MOV DX,STATUS
```

```
    IN AL,DX
    AND  AL,TXRDY
    RET
STATIN2  ENDP
PIN   PROC NEAR
    MOV DX,DATA1
    IN AL,DX
    RET
PIN   ENDP
CRLF   PROC NEAR
    PUSH AX
    MOV   AL,0DH                ;CARRIAGE RETURN
    CALL OUTPUT
    MOV   AL,0AH                ;LINE FEED
    CALL OUTPUT
    POP AX
    RET
CRLF  ENDP
OUTPUT   PROC NEAR
    PUSH ES                     ;SAVE THE ES,
    PUSH BX                     ;BX
    PUSH DX                     ;DX,
    PUSH CX                     ;AND CX REGISTERS
    LAHF
```

```
    PUSH AX
    MCV  AH,CONOUT          ;PEINT TO SCREEN
    MCV  DL,AL              ;PUT THE ACCUMULATOR IN 'DI' RGSTR
    CALL BDOS
    PCP  AX
    SAHF
    POP  CX                 ;RETURN THE CX,
    POP  DX                 ;DX,
    PCP  BX                 ;BX,
    POP  ES                 ;AND ES REGISTERS
    RET
OUTPUT   ENDP
BDOS     PROC NEAR
    PUSH ES                 ;SAVE THE ES
    PUSH BX                 ;BX,
    PUSH DX                 ;DX,
    PUSH CX                 ;AND CX REGISTERS
    INT 21H                 ;EXECUTE
    POP  CX                 ;RETURN THE CX,
    POP  DX                 ;DX,
    PCP  BX                 ;BX,
    PCP ES                  ;AND ES REGISTERS
    RET
BDOS  ENDP
```

```
CPM: RET                        ;FAR RETURN
ERROR1: MOV DX,OFFSET ERR1      ;BAUDRATE ERROR
    MOV AH,09H                  ;PRINT STRING TO SCREEN
    CALL BDOS
    RET                         ;FAR RETURN
START ENDP
CODE  ENDS
    END   START
```

NOTE: This version of MICROLAN was obtained by converting the CP/M-86 version to meet the requirements of the MS.DOS operating system. We did not attempt to streamline the procedure or efficiency of operation for this IBM compatible version.

# APPENDIX F

## MICROLAN'S ON SCREEN MESSAGES

| | |
|---|---|
| BAUDMSG | SELECT BAUD RATE |
| | 1 = 300 BAUD |
| | 2 = 600 BAUD |
| | 3 =1200 BAUD |
| | 4 =2400 BAUD |
| | 5 =4800 BAUD |
| | 6 =9600 BAUD |
| ERR1 | BAUDRATE OUT OF RANGE |
| RIGHTS | MICROLAN VERSION 2.1 |
| | COPYRIGHT (C) 1985 ROGER D. JASKOT AND HAROLD W. HENRY |
| WELCUM | WELCOME! |
| | YOU ARE NOW ENTERING THE TRANSFER ZONE! |
| INSTRC | ENTER AN S FOR TRANSMIT MODE, AN R FOR RECEIVE MODE, |
| | OR AN X TO EXIT. |
| ENTER | ENTER NAME OF FILE TO BE SENT. IF THE FILE IS ON |
| | A DISK IN ANOTHER DRIVE, ENTER IN THE FORMAT: |
| | B:FILENAME.FILETYPE |
| ERMSG | ENTER FILENAME AGAIN. END WITH <CR> |

| | |
|---|---|
| WCHDSK | WRITE FILE TO WHICH DISK DRIVE? ENTER AN A FOR A DRIVE,<br>A B FOR B DRIVE,... OR PRESS RETURN FOR DEFAULT DRIVE. |
| RXMODE | IN RECEIVE MODE. |
| NOMSG | NO FILE TRANSFER. RETURNING TO CPM. |
| FNFDMSG | FILE DOES NOT EXIST, RETURNING TO CPM. |
| FNDMSG | FILE ALREADY EXISTS. RETURNING TO CPM. |
| HASFILE | RXING MICRO HAS FILE ALREADY, GOING TO CPM. |
| RXING1 | CONNECTION MADE. |
| TXING1 | TRANSMITTING FILE. |
| FULLMSG | DISK FULL. FILE TRANSFER INCOMPLETE. |
| FULMSG | RXING MICRO DISK FULL. RETURNING TO CPM. |
| ABRTMSG | XMITING MICRO ABORTED FILE TRANSFER.<br>PLEASE ERASE FILENAME FROM YOUR DIRECTORY. |
| EOFMSG | FILE TRANSMISSION COMPLETED. |

## LIST OF REFERENCES


1. Rosner, R. D., *Packet Switching Tomorrows Communications Today*, Lifetime Learning Publications, 1982.

2. Jordan, Larry E. and Churchill, Bruce, *Communications and Networking for the IBM PC*, Robert J. Brady Company, 1983.

3. Akscyn, Robert M. and McCracken, Donald L., *The ZOG Approach to Database Management*, Computer Science Department, Carnegie-Mellon University, March, 1984.

4. *Operational and Organizational Plan for the Command Post Automated Staff Support (CPASS) System*, Command, Control, Communication and U.S. Army Combined Arms Combat Developments Activity, January 30, 1984.

5. Southeastern Center for Electrical Engineering Education Inc, *Internal Communications for the Csan HTACC (Hardened Tactical Air Control Center)*, by R. VanSlyke, G. Herskowitz, and A. Kershenbaum, October, 1983.

6. Mitre Corporation, *PENTANET--A Report of the United States Air Force Technical for a Pentagon-wide Local Area Network*, by R.A. Creamer and C.E. Dolberg, March, 1983.

7. Mitre Corporation, *Base Support Communications: General System Description*, by J.W. Guppy, C.J. Ludinsky, and J.P. Worthley, March, 1983.

8. Tanenbaum, A. S., *Computer Networks*, Prentice-Hall, 1981.

# BIBLIOGRAPHY

Akscyn, Robert M. and McCracken, Donald L., ZOG and the USS Carl Vinson: Lessons in System Development, Computer Science Department Carnegie-Mellon University, March, 1984.

Metcalfe, Robert M. and Meyer, N. Dean, "Moving Ahead With Local Nets", Computerworld, 1984.

# INITIAL DISTRIBUTION LIST

| | | No. Copies |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, Virginia 22314 | 2 |
| 2. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, California 93943 | 2 |
| 3. | Lt Roger D. Jasket<br>4428 Clemsford Drive<br>Virginia Beach, Virginia 23456 | 1 |
| 4. | HQ TAC/SIXXP<br>ATTN: Capt Harold W. Henry<br>Langley AFB, Virginia 23665 | 2 |
| 5. | IMAF IMSO<br>BLDG 16104<br>ATTN: MSgt Hefner<br>Camp Pendleton, Ca 92055 | 1 |
| 6. | TAFIG/IIA<br>ATTN: Capt Chuck<br>Langley AFB, Virginia 23665 | 1 |
| 7. | Professor Michael G. Sovereign<br>Department of Operations Research, Code 74<br>Naval Postgraduate School<br>Monterey, California 93943 | 10 |
| 8. | Professor Gordon E. Latta, Chairman<br>Department of Mathematics, Code 53<br>Naval Postgraduate School<br>Monterey, California 93943 | 1 |
| 9. | LtCol John T. Malokas<br>ATTN: Code 39<br>Naval Postgraduate School<br>Monterey, California 93943 | 1 |
| 10. | AFIT/CIRS<br>Wright-Patterson AFB, Ohio 45433 | 1 |